SuperDataScience

NEURAL NETWORKS IN PYTHON

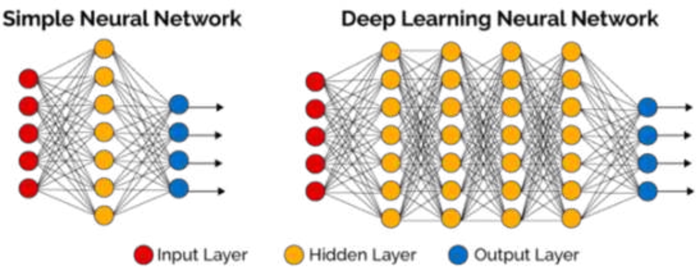
NEURAL NETWORKS IN PYTHON

1. Part 1
   - Biological fundamentals
   - Single layer perceptron
2. Part 2
   - Multi-layer perceptron
3. Part 3
   - Pybrain
   - Sklearn
   - TensorFlow
   - PyTorch

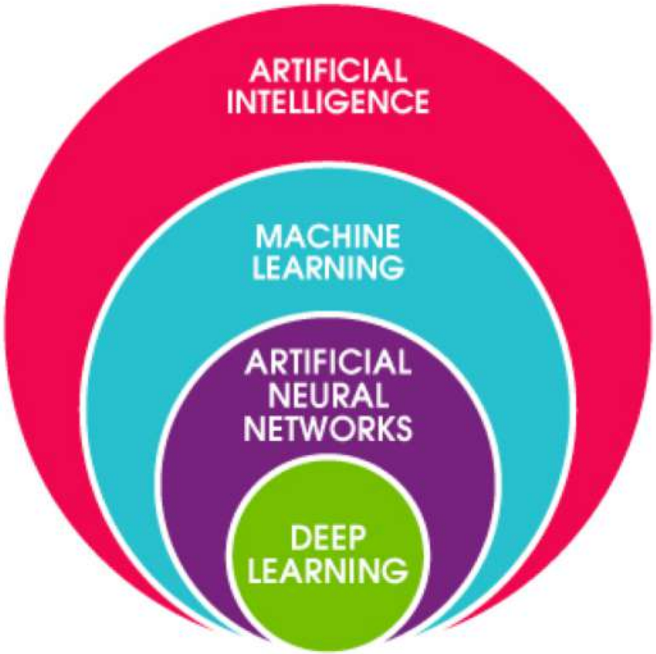
SuperDataScience

NEURAL
NETWORKS

1. WHAT ARE NEURAL NETWORKS?



Source: https://www.meetup.com/pt-BR/Deep-Learning-for-Sciences-Engineering-and-Arts/events/257483663/



Source: https://elearningindustry.com/artificial-intelligence-will-shape-elearning

Artificial intelligence

Expert systems
Computer vision
Genetic algorithms
Natural language processing
Fuzzy logic
Case based reasoning
Multi-agent systems
Machine learning

Machine learning

Naïve Bayes
Decision trees
SVM
K-NN
Rules
Artificial neural networks

2. WHAT ARE THE APPLICATIONS OF NEURAL NETWORKS?

Source: https://www.livemint.com/Technology/T4CcI7ZfnyWiWKvoOPM5mK/Your-face-can-unlock-your-smartphone-But-is-it-safe.html

Source: https://www.vero.com.br/ricardo-cancela-veiculos-autonomos-uma-realidade-proxima/empty-cockpit-of-autonomous-car-hudhead-up-display-and-digital-speedometer-self-driving-vehicle/

Source: https://towardsdatascience.com/feature-engineering-in-stock-market-prediction-quantifying-market-vs-fundamentals-3895ab9f311f

Source: https://dreamdeeply.com/

Source: https://www.endpoint-translation.com/Blog?id=1785129625&fbclid=IwAR1dQaEb73swW0iEVdgmDrA5a4Ni_PGZ4kNb-3UiMm4Nitd0Ureq32uagXg

Source: https://www.theverge.com/tldr/2019/2/15/18226005/ai-generated-fake-people-portraits-thispersondoesnotexist-stylegan

3. WHY STUDY NEURAL NETWORKS?

PLAN OF ATTACK – SINGLE LAYER PERCEPTRON

1. Neural networks applications
2. Biological fundamentals
3. Artificial neuron (perceptron)
4. Implementation of a perceptron from scratch using Python and Numpy

NEURAL NETWORKS APPLICATIONS

Source: https://www.livemint.com/Technology/T4CcI7ZfnyWiWKyoOPM5mK/Your-face-can-unlock-your-smartphone-But-is-it-safe.html

Source: https://www.vero.com.br/ricardo-cancela-veiculos-autonomos-uma-realidade-proxima/empty-cockpit-of-autonomous-car-hudhead-up-display-and-digital-speedometer-self-driving-vehicle/

Source: https://www.biosciencetoday.co.uk/artificial-neural-networks-working-with-image-guided-therapies-to-improve-heart-disease-treatment/

NEURAL NETWORKS APPLICATIONS

Source: https://towardsdatascience.com/feature-engineering-in-stock-market-prediction-quantifying-market-vs-fundamentals-3895ab9f311f

Source: https://www.theverge.com/tldr/2019/2/15/18226005/ai-generated-fake-people-portraits-thispersondoesnotexist-stylegan

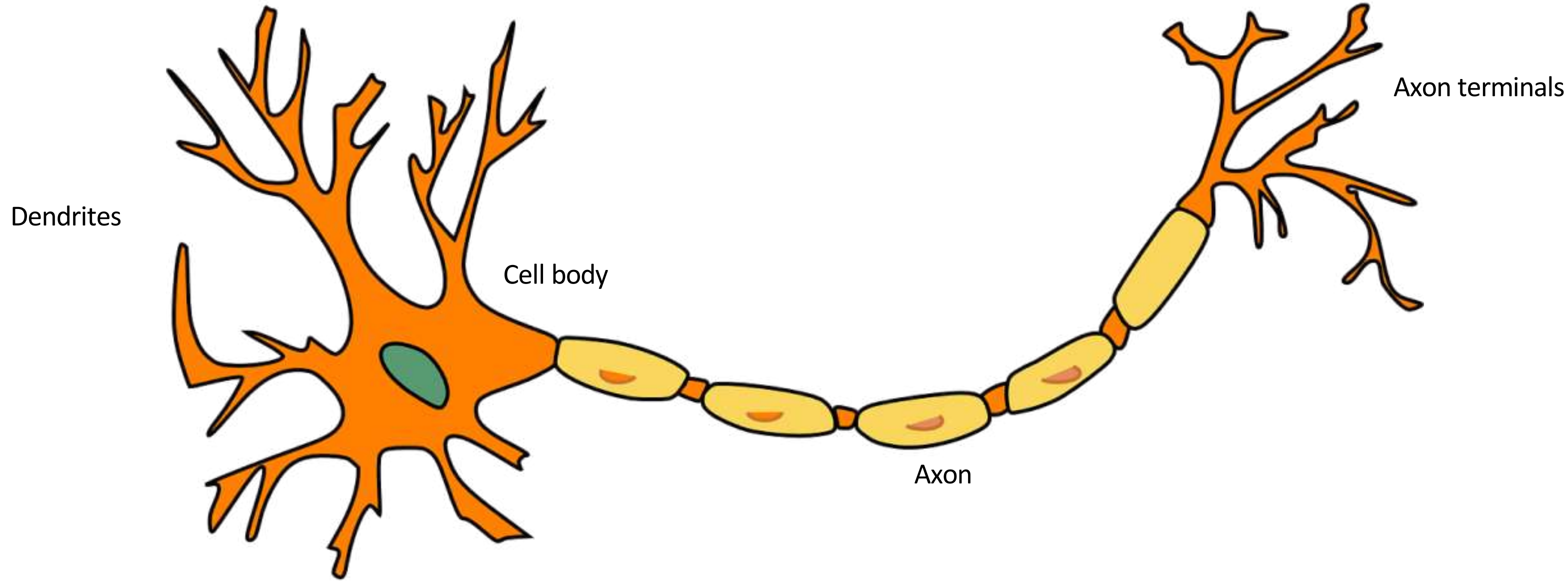
Source: https://www.endpoint-translation.com/Blog?id=1785129625&fbclid=IwAR1dQaEb73swW0iEVdgmDrA5a4Ni_PGZ4kNb-3UiMm4Nitd0Ureq32uagXg

Source: https://dreamdeeply.com/

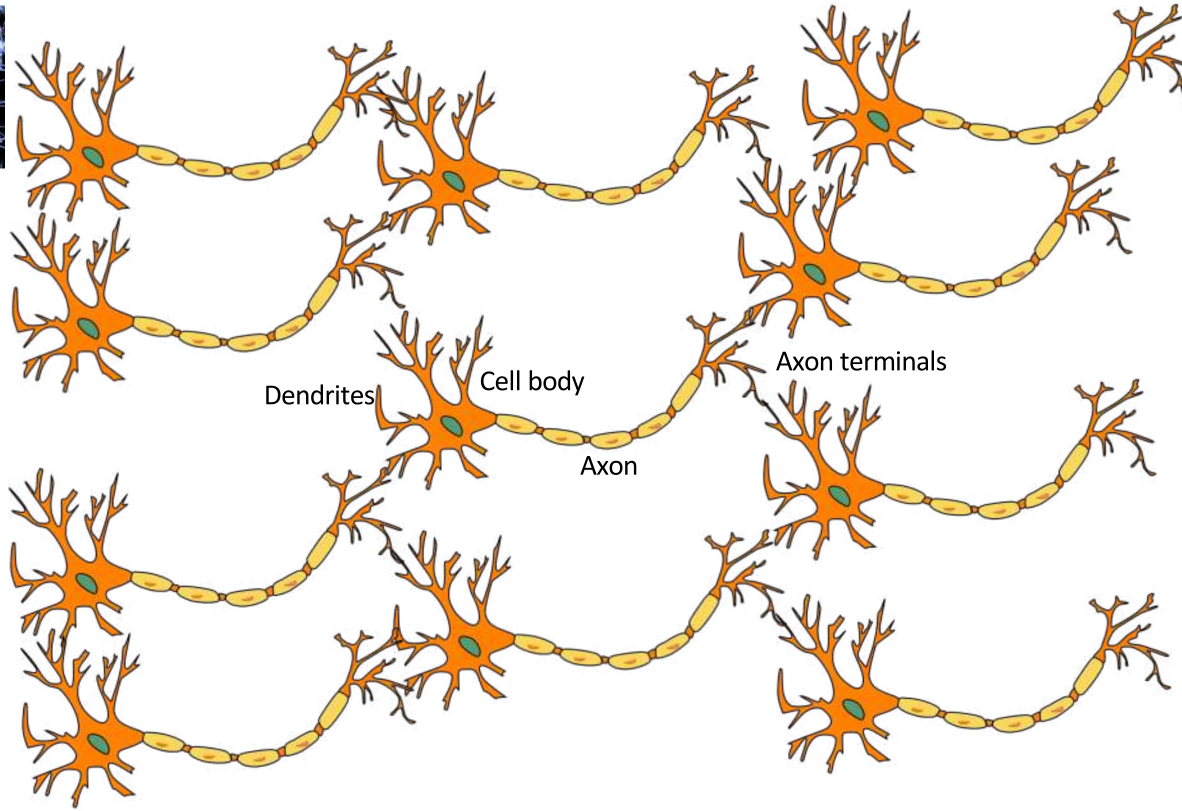
python

BIOLOGICAL FUNDAMENTALS

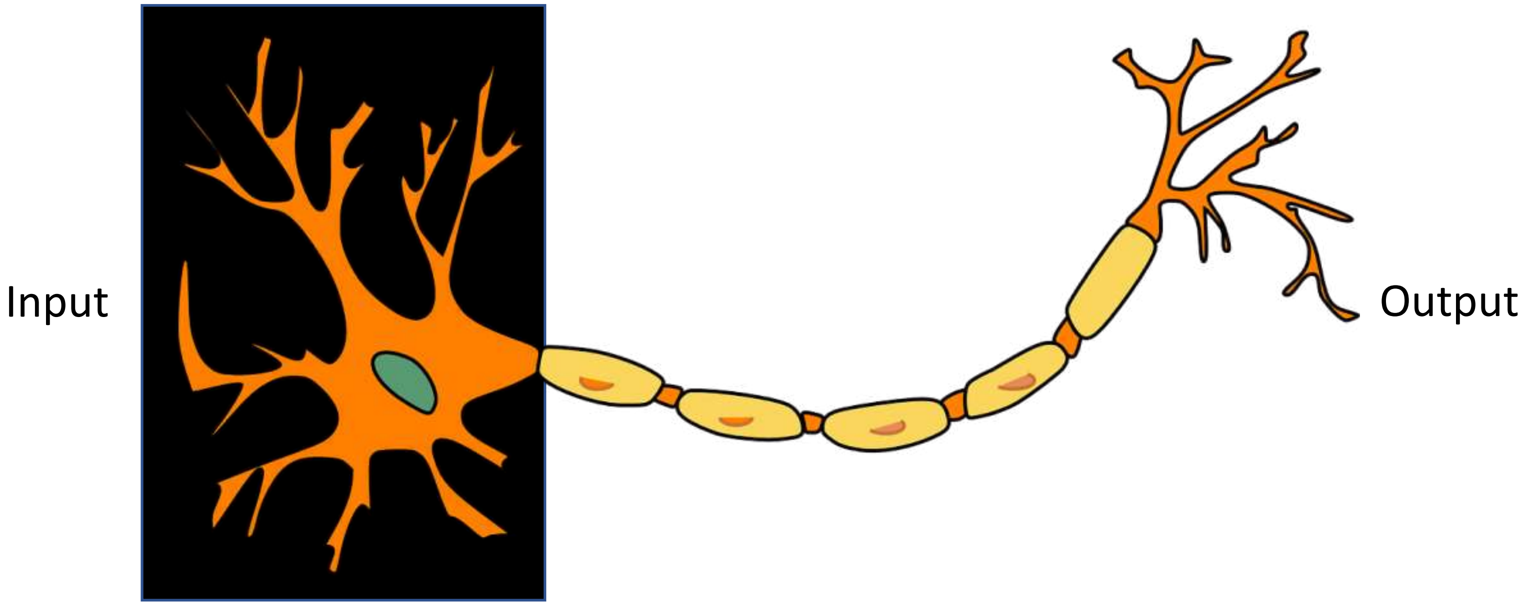
BIOLOGICAL FUNDAMENTALS



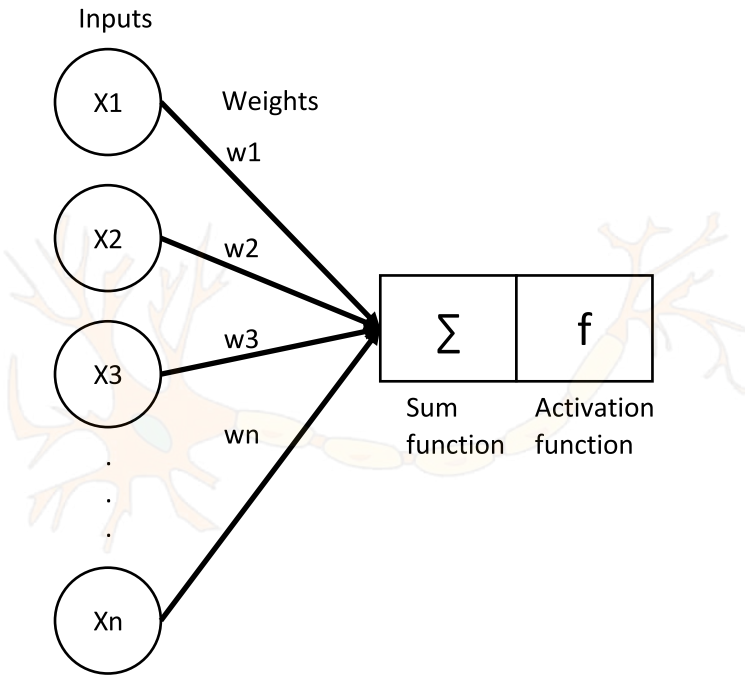
BIOLOGICAL FUNDAMENTALS



ARTIFICIAL NEURON



ARTIFICIAL NEURON

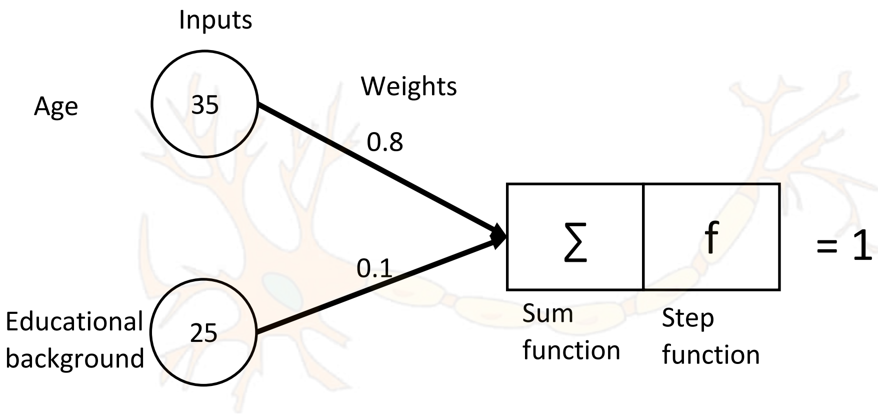


$$sum = \sum_{i=1}^{n} xi * wi$$

X1 * w1 + X2 * w2 + X3 * w3

python

PERCEPTRON



$$sum = \sum_{i=1}^{n} xi \ * wi$$

$$sum = (35 * 0.8) + (25 * 0.1)$$

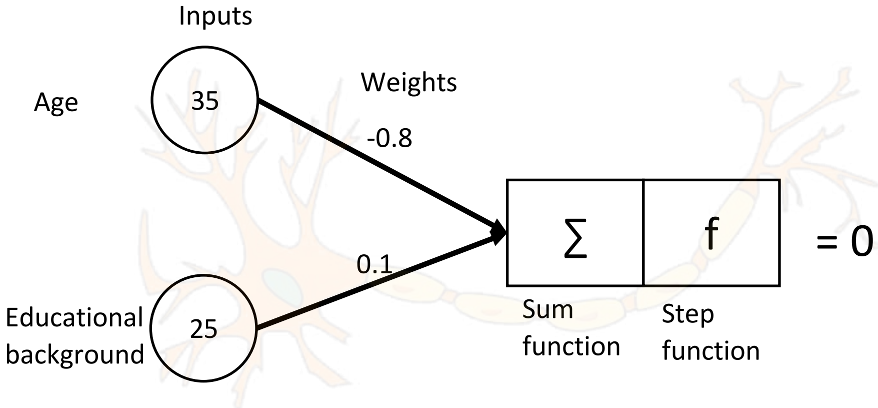
$$sum = 28 + 2.5$$

$$sum = 30.5$$

Greater or equal to 1 = 1
Otherwise = 0

“All or nothing” representation

PERCEPTRON



$$sum = \sum_{i=1}^{n} xi * wi$$

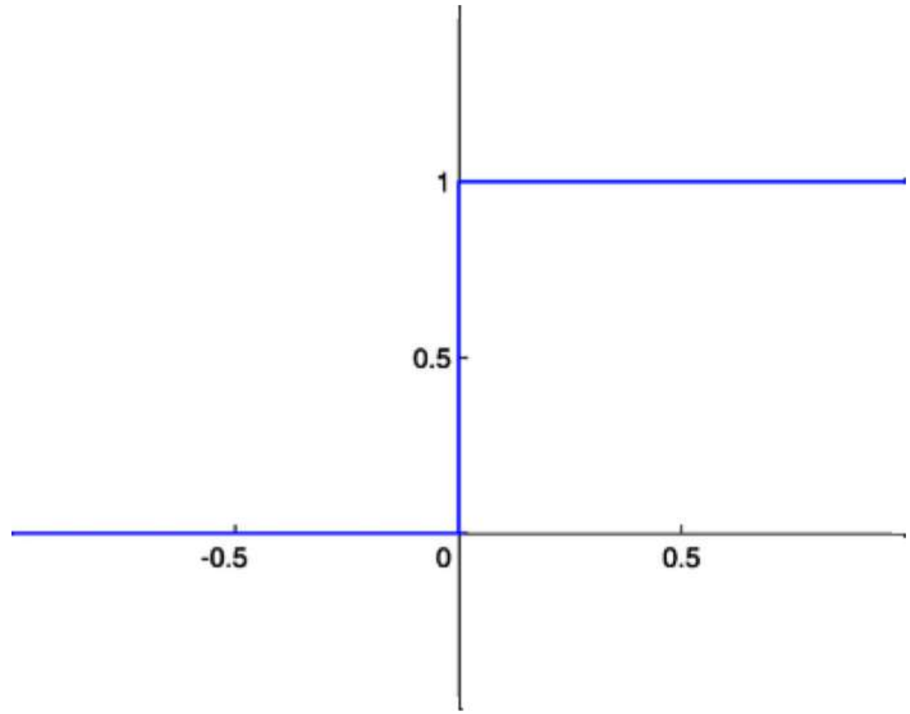
$sum = (35 * -0.8) + (25 * 0.1)$

$sum = -28 + 2.5$

$sum = -25.5$

Greater or equal to 1 = 1
Otherwise = 0

STEP FUNCTION



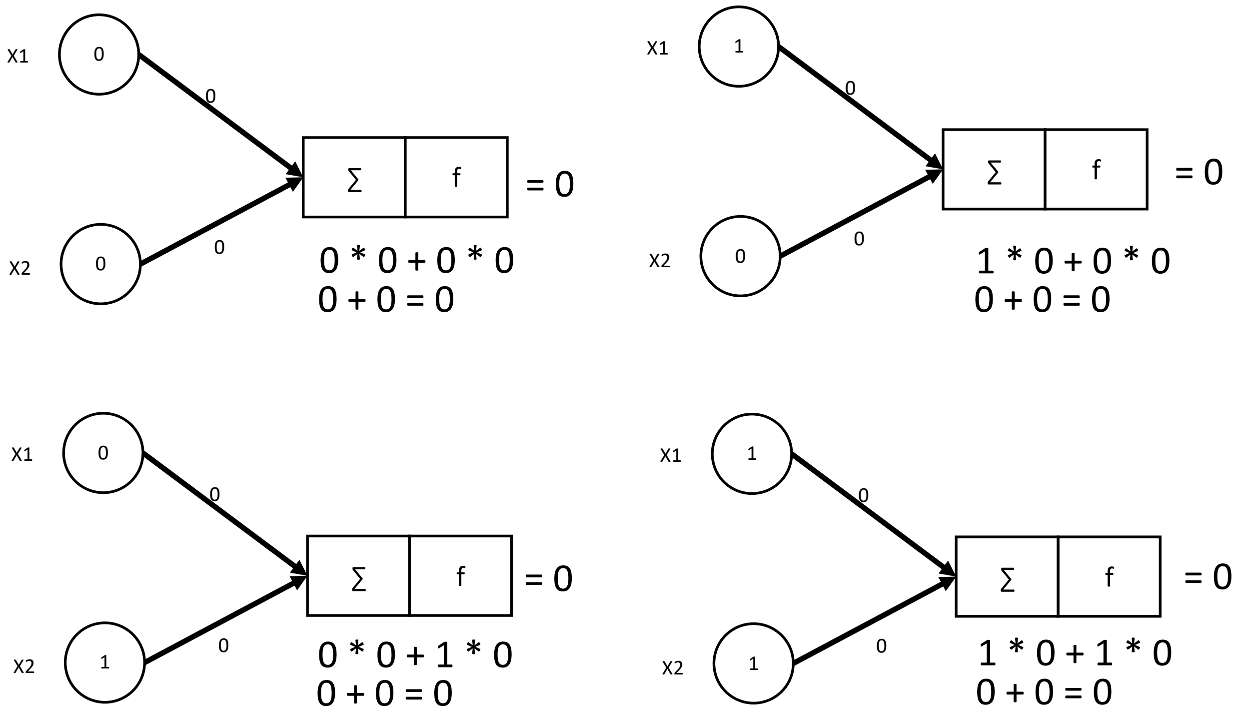
PERCEPTRON

- Positive weight – exciting synapse
- Negative weight – inhibitory synapse
- Weights are the synapses
- Weights amplify or reduce the input signal
- The knowledge of a neural network is the weights

"AND" OPERATOR

X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1

“AND” OPERATOR



X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

75%

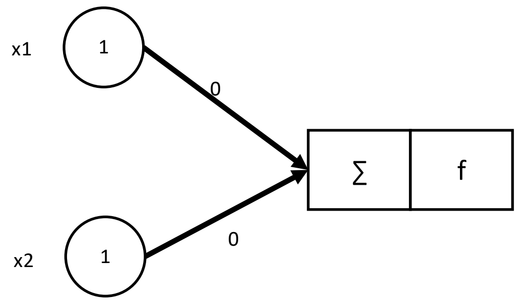
weight (n + 1) = weight(n) + (learning_rate * input * error)

python

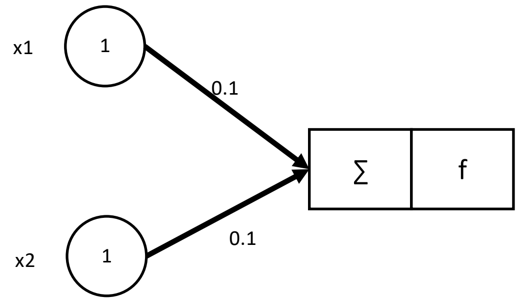
"AND" OPERATOR

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

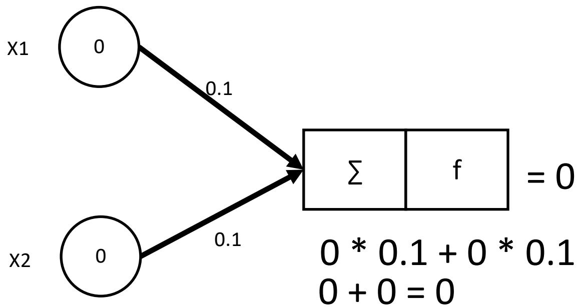


weight (n + 1) = weight(n) + (learning_rate * input * error)
weight (n + 1) = 0 + (0.1 * 1 * 1)
weight (n + 1) = 0.1

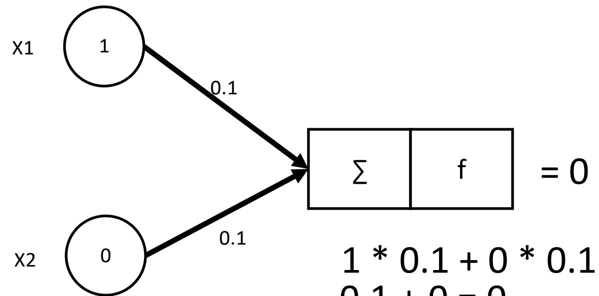


weight (n + 1) = weight(n) + (learning_rate * input * error)
weight (n + 1) = 0 + (0.1 * 1 * 1)
weight (n + 1) = 0.1

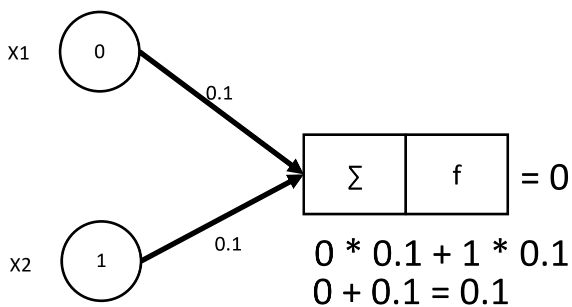
“AND” OPERATOR



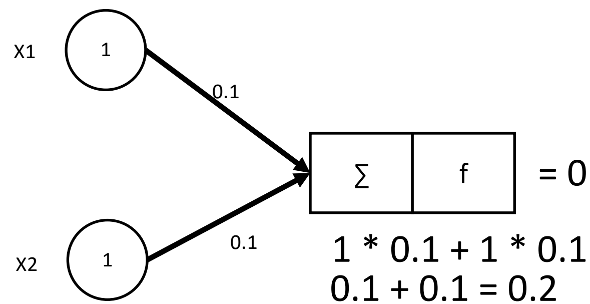
0 * 0.1 + 0 * 0.1
0 + 0 = 0



1 * 0.1 + 0 * 0.1
0.1 + 0 = 0



0 * 0.1 + 1 * 0.1
0 + 0.1 = 0.1



1 * 0.1 + 1 * 0.1
0.1 + 0.1 = 0.2

X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

75%

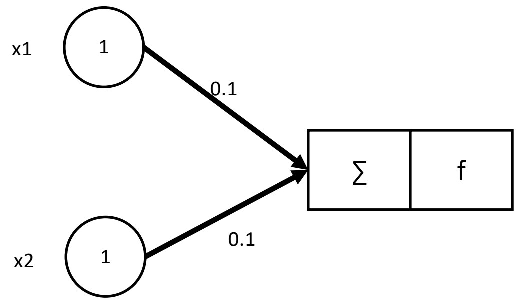
python

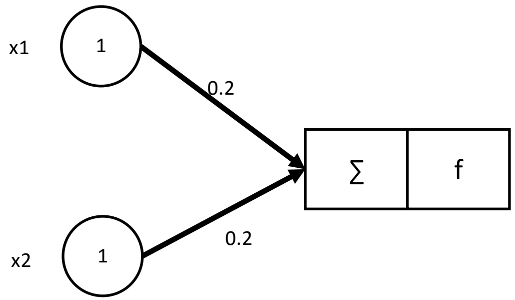
"AND" OPERATOR

weight (n + 1) = weight(n) + (learning_rate * input * error)
weight (n + 1) = 0.1 + (0.1 * 1 * 1)
weight (n + 1) = 0.2

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

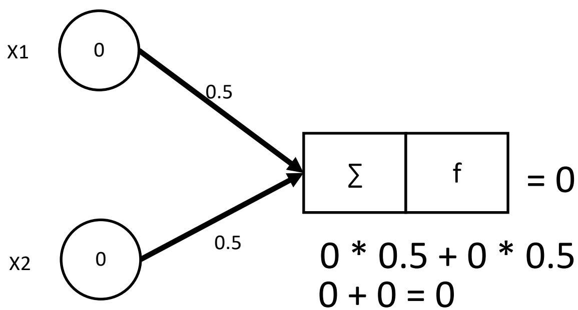


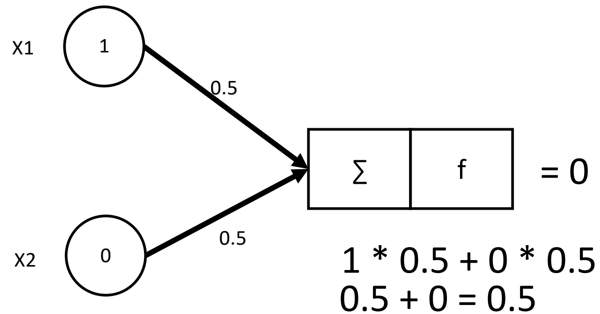


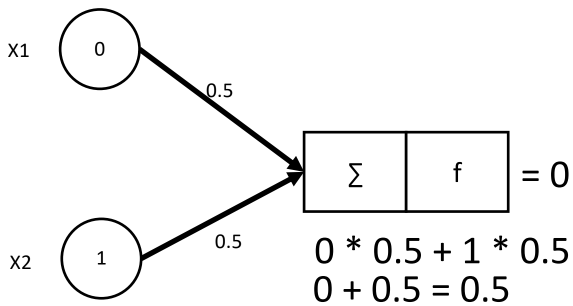
weight (n + 1) = weight(n) + (learning_rate * input * error)
weight (n + 1) = 0.1 + (0.1 * 1 * 1)
weight (n + 1) = 0.2

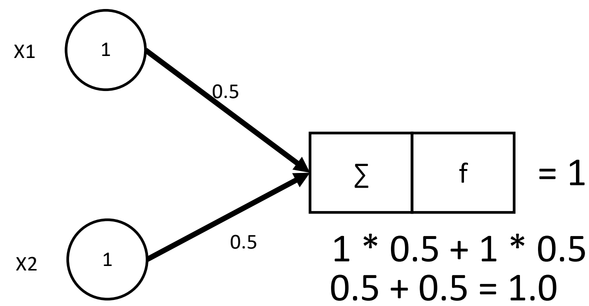
python

“AND” OPERATOR









X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	1	0

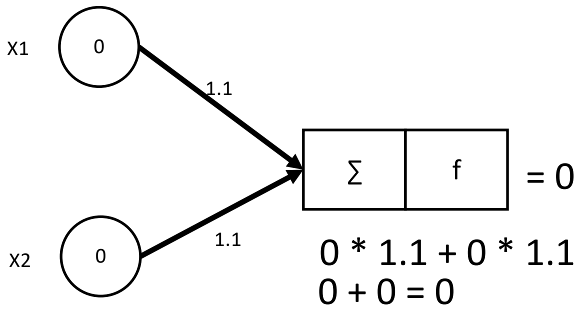
100%

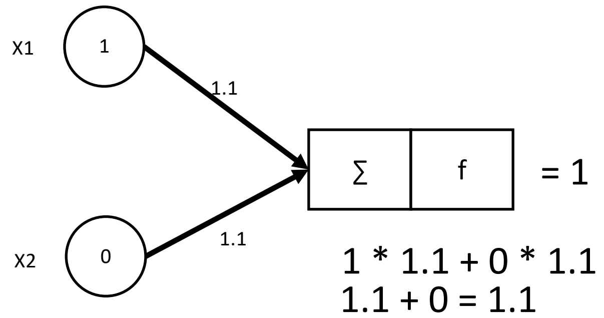
python

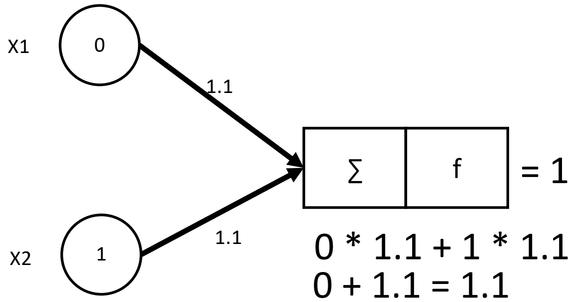
BASIC ALGORITHM

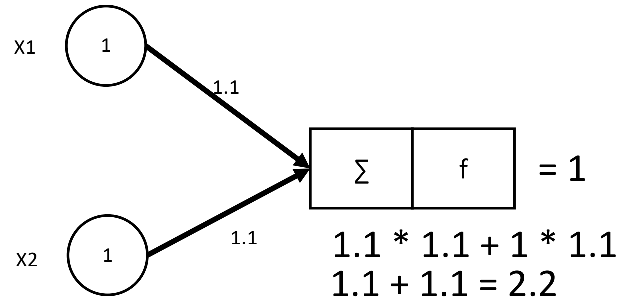
```
While error <> 0
   For each row
      Calculate output
      Calculate error (correct - prediction)
      If error > 0
            For each weight
                  Update the weights
```

“OR” OPERATOR









X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	1

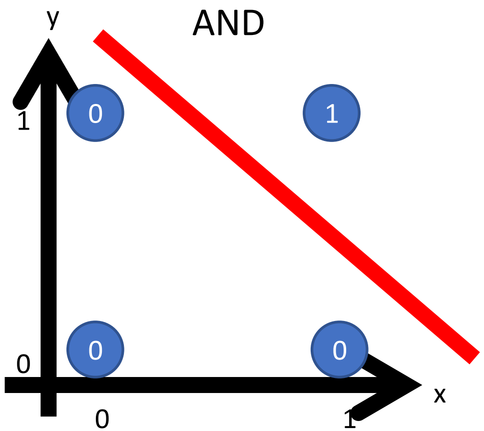
error = correct - prediction

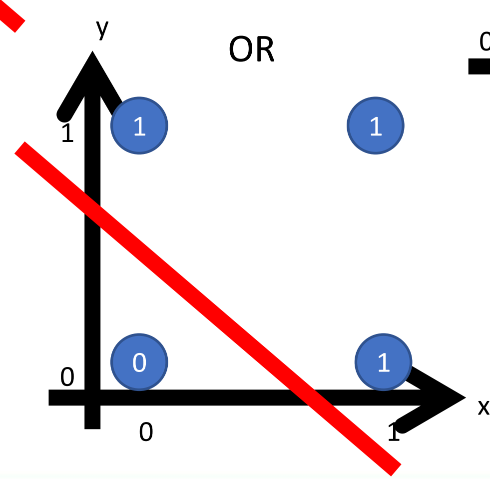
Class	Prediction	Error
0	0	0
1	1	0
1	1	0
1	1	0

100%

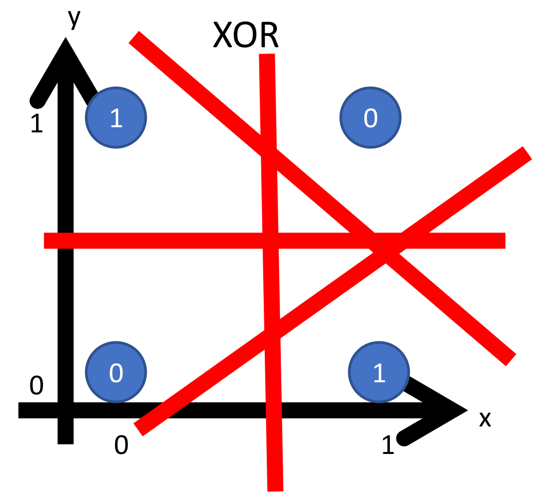
python

“XOR” OPERATOR





Linearly separable

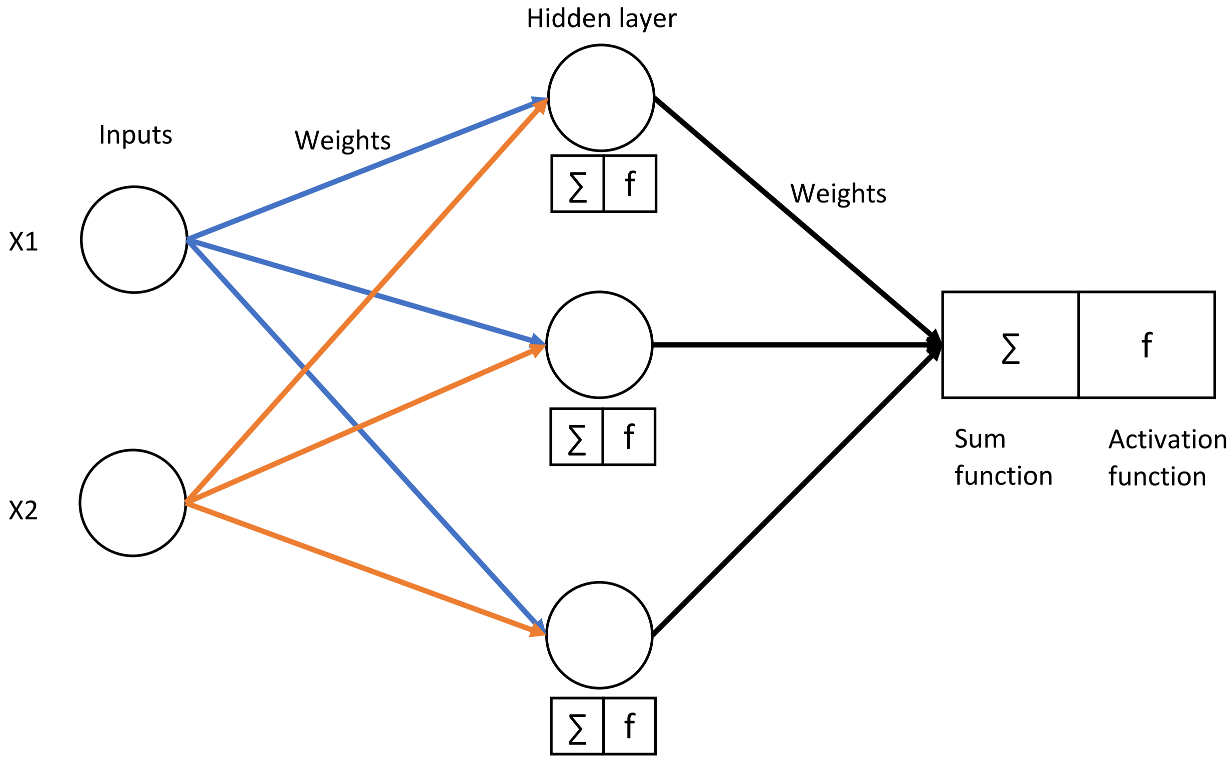


Non-linearly separable

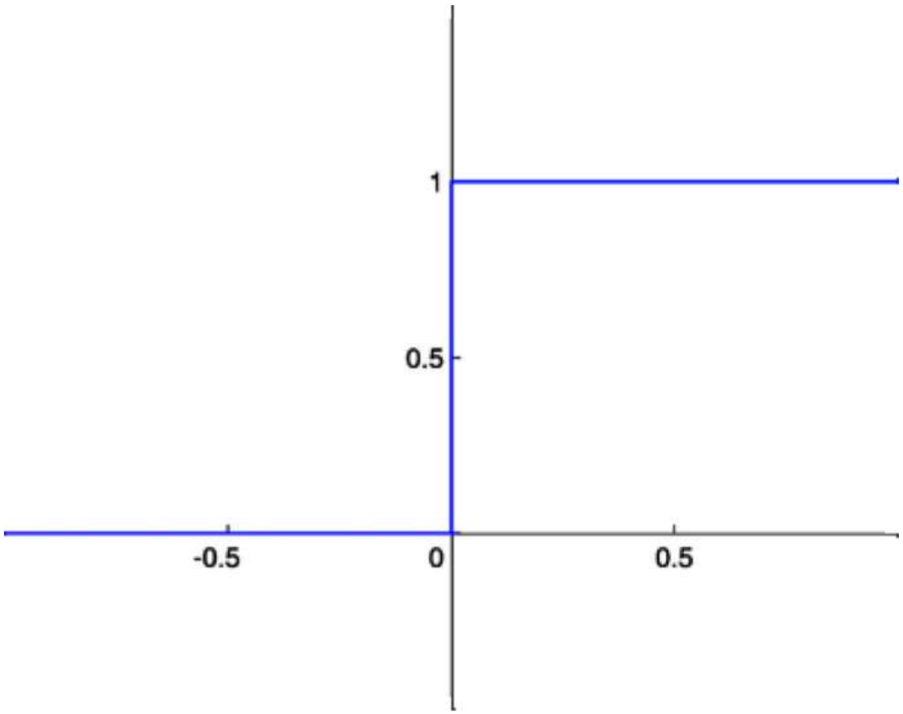
PLAN OF ATTACK – MULTI-LAYER PERCEPTRON

1. Single layer and multi-layer
2. Activation functions
3. Weight update (XOR operator)
4. Error functions
5. Gradient descent
6. Backpropagation
7. Implementation of a multi-layer perceptron from scratch using Python and Numpy

MULTI-LAYER PERCEPTRON

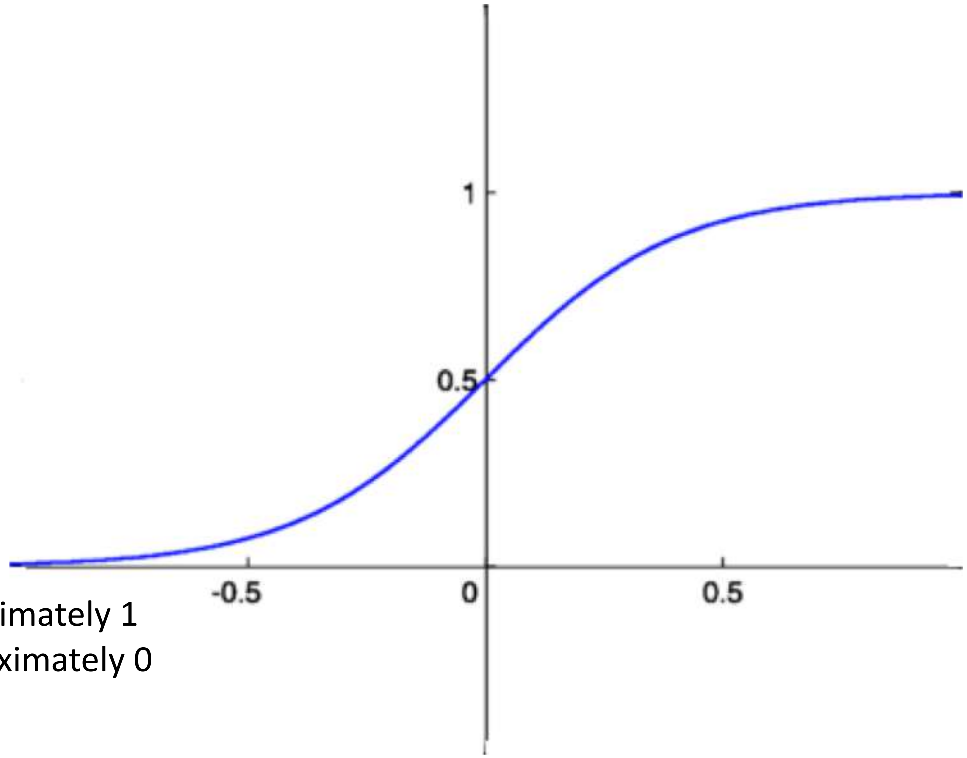


STEP FUNCTION



SIGMOID FUNCTION

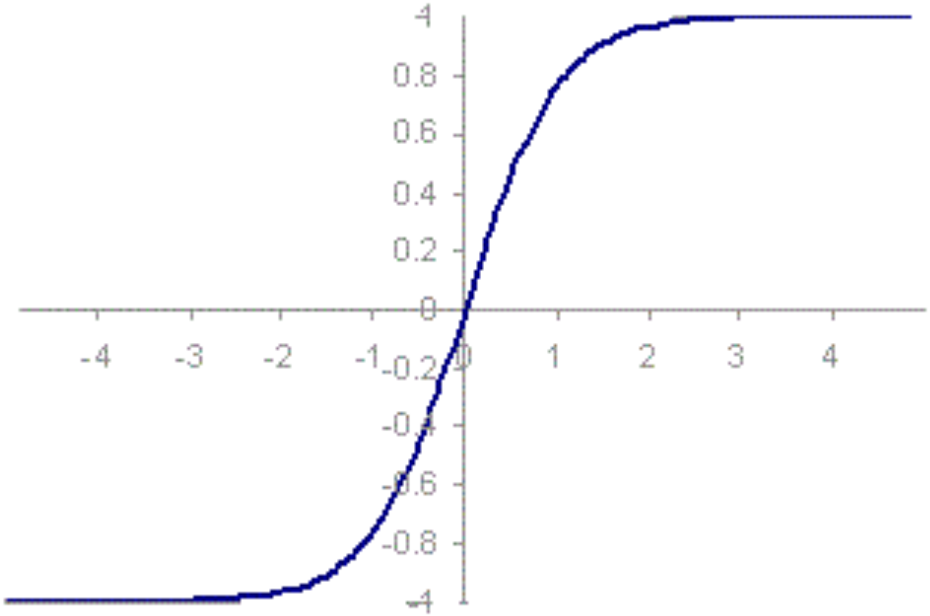
$$y = \frac{1}{1 + e^{-x}}$$



If X is high, the value is approximately 1
If X is small, the value is approximately 0

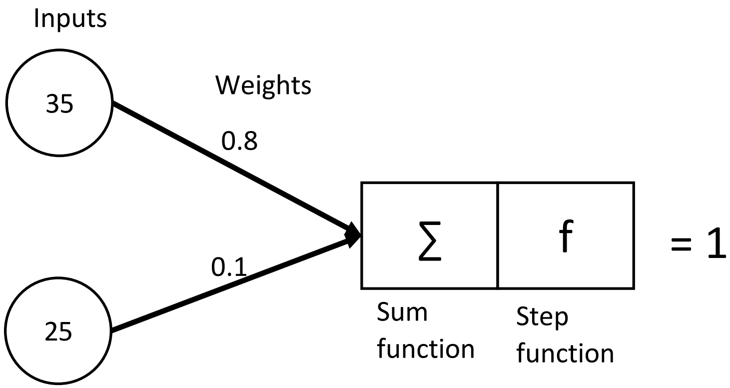
HYPERBOLIC TANGENT FUNCTION

$$Y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



python

STEP FUNCTION

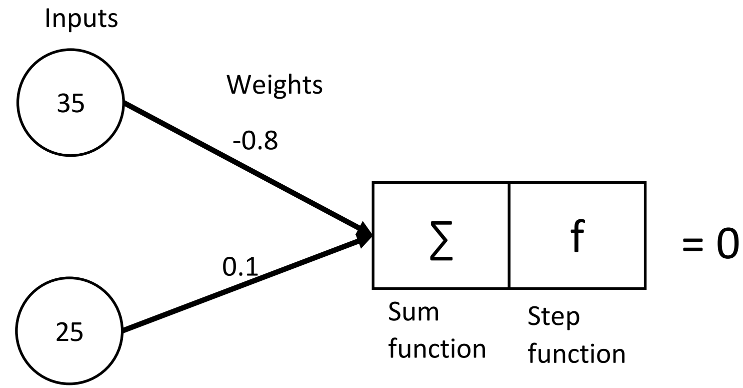


$$sum = \sum_{i=1}^{n} xi * wi$$

$sum = (35 * 0.8) + (25 * 0.1)$

$sum = 28 + 2.5$

$sum = 30.5$



$$sum = \sum_{i=1}^{n} xi * wi$$

$sum = (35 * -0.8) + (25 * 0.1)$

$sum = -28 + 2.5$

$sum = -25.5$

python

“XOR” OPERATOR

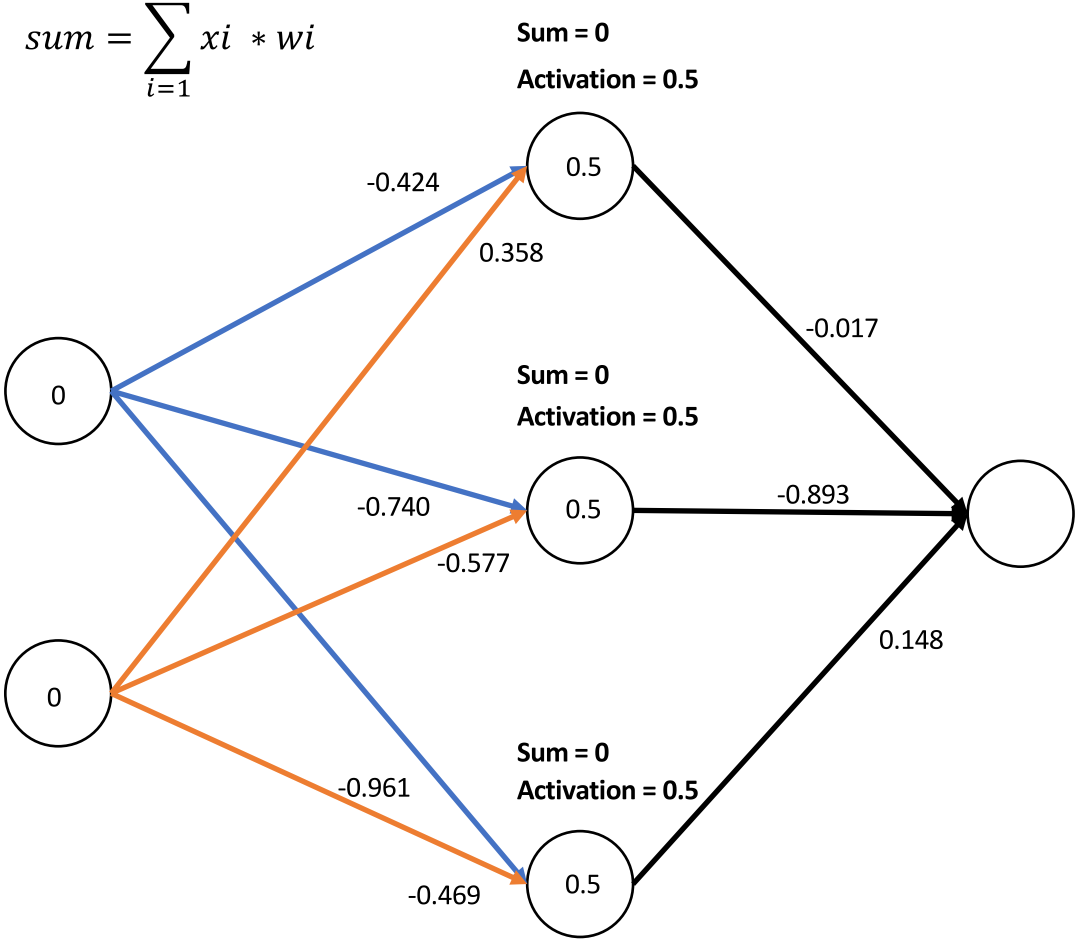
X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

INPUT LAYER TO HIDDEN LAYER

$$sum = \sum_{i=1}^{n} xi \ * wi$$

$$y = \frac{1}{1 + e^{-x}}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



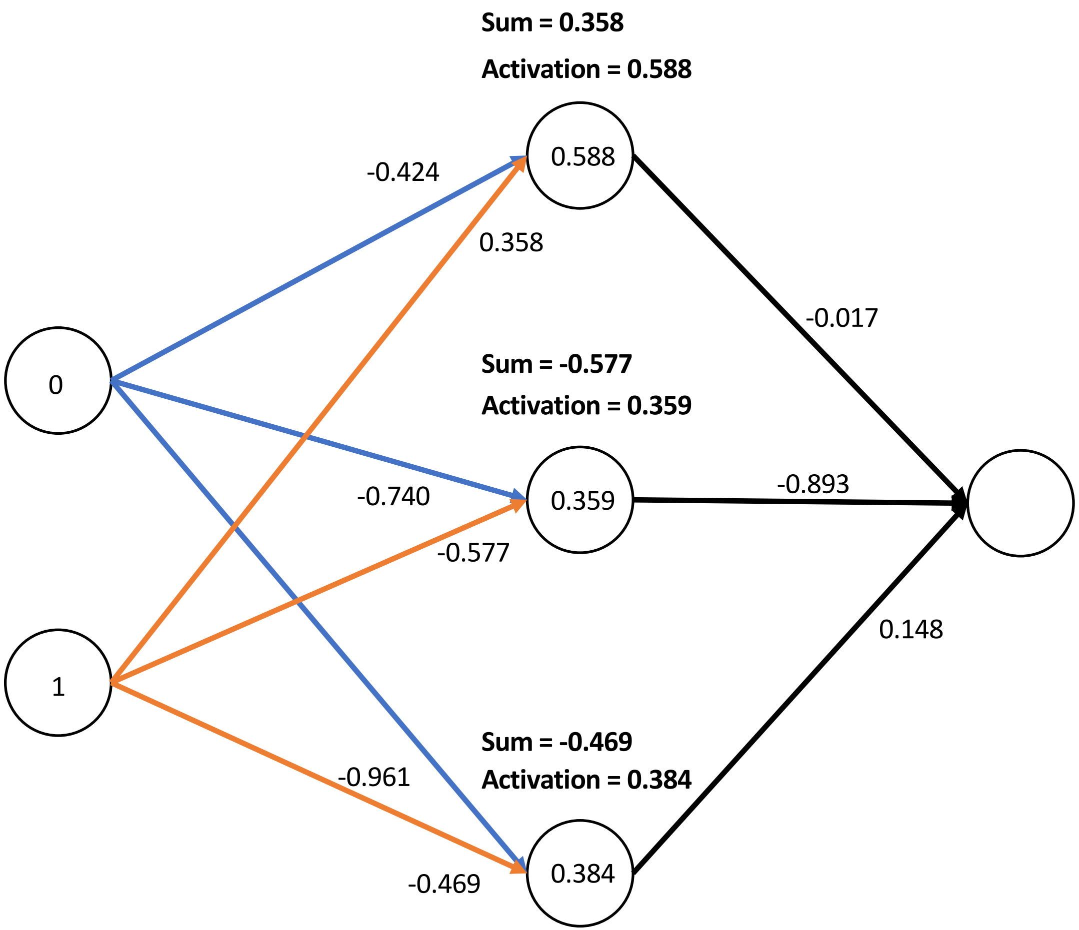
0 * (-0.424) + 0 * 0.358 = 0

0 * (-0.740) + 0 * (-0.577) = 0

0 * (-0.961) + 0 * (-0.469) = 0

INPUT LAYER TO HIDDEN LAYER

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



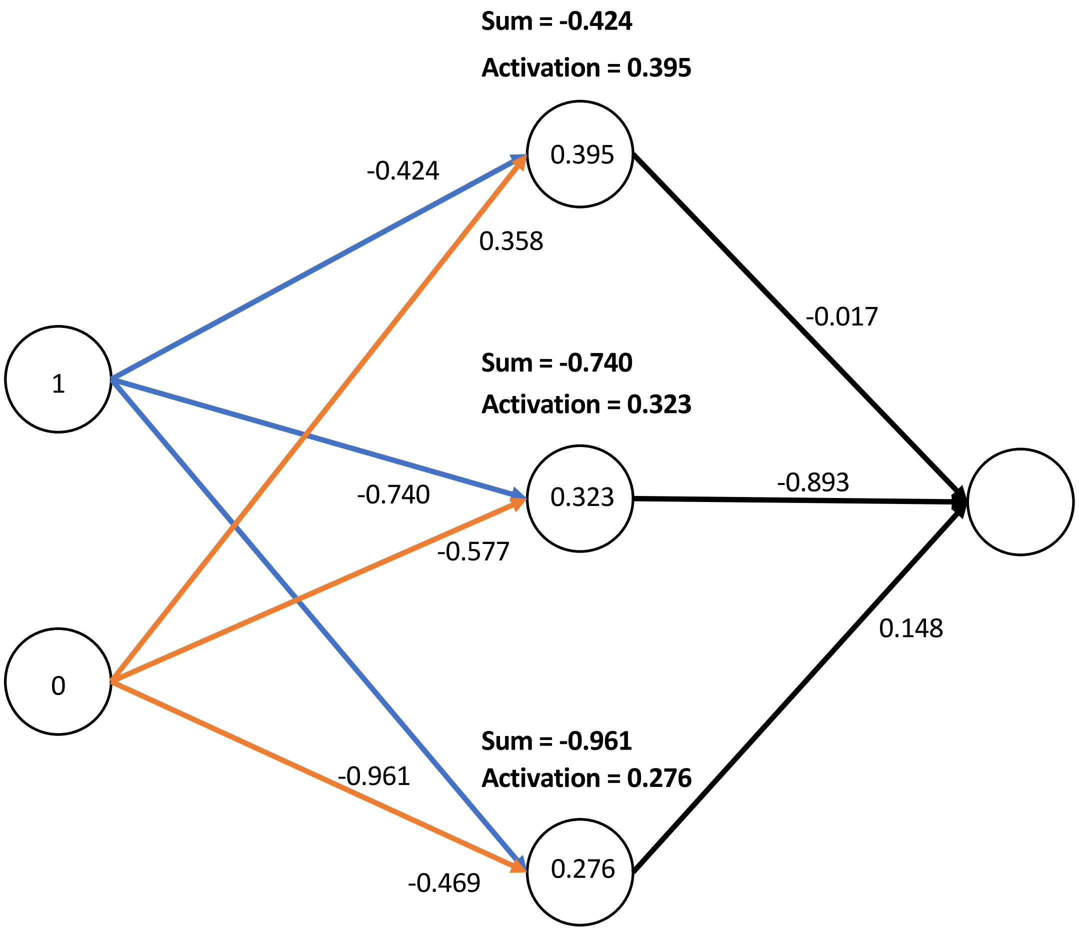
0 * (-0.424) + 1 * 0.358 = 0.358

0 * (-0.740) + 1 * (-0.577) = -0.577

0 * (-0.961) + 1 * (-0.469) = -0.469

INPUT LAYER TO HIDDEN LAYER

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



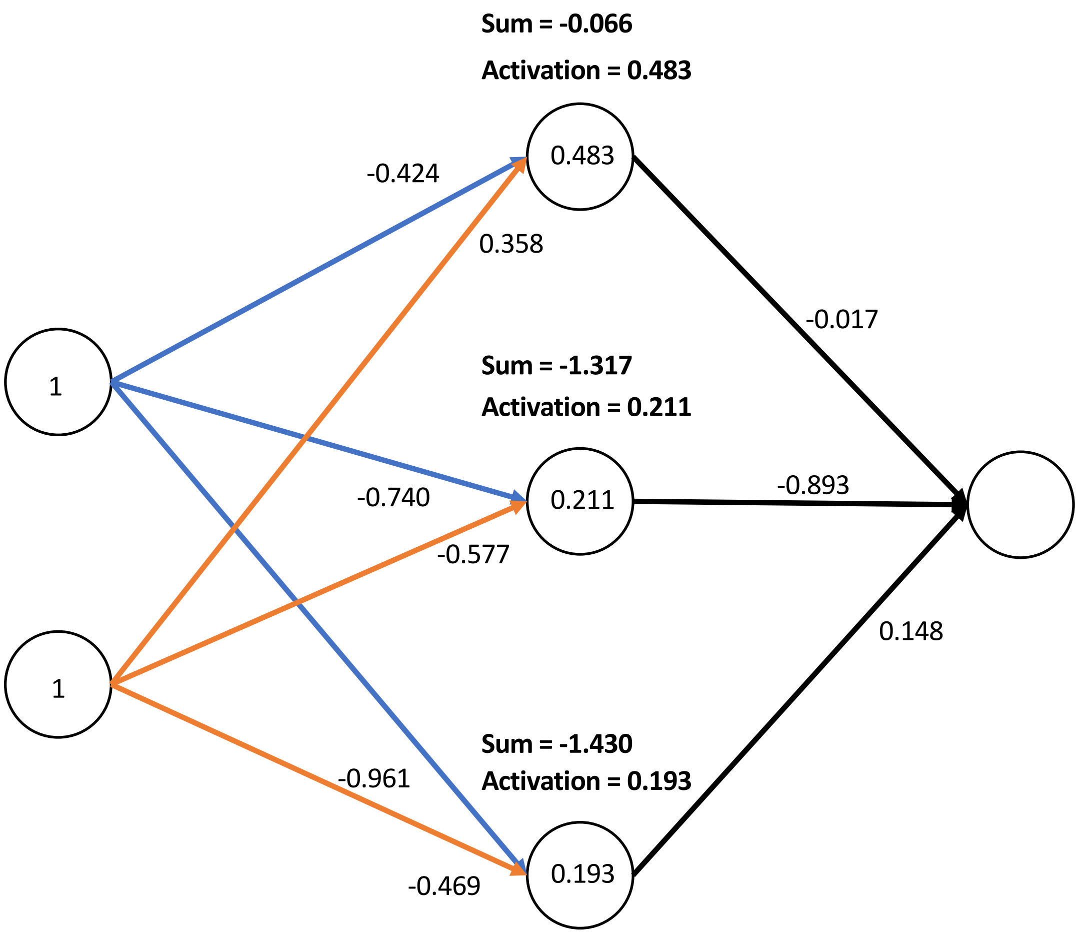
1 * (-0.424) + 0 * 0.358 = -0.424

1 * (-0.740) + 0 * (-0.577) = -0.740

1 * (-0.961) + 0 * (-0.469) = -0.961

INPUT LAYER TO HIDDEN LAYER

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



$$1 * (-0.424) + 1 * 0.358 = -0.066$$

$$1 * (-0.740) + 1 * (-0.577) = -1.317$$

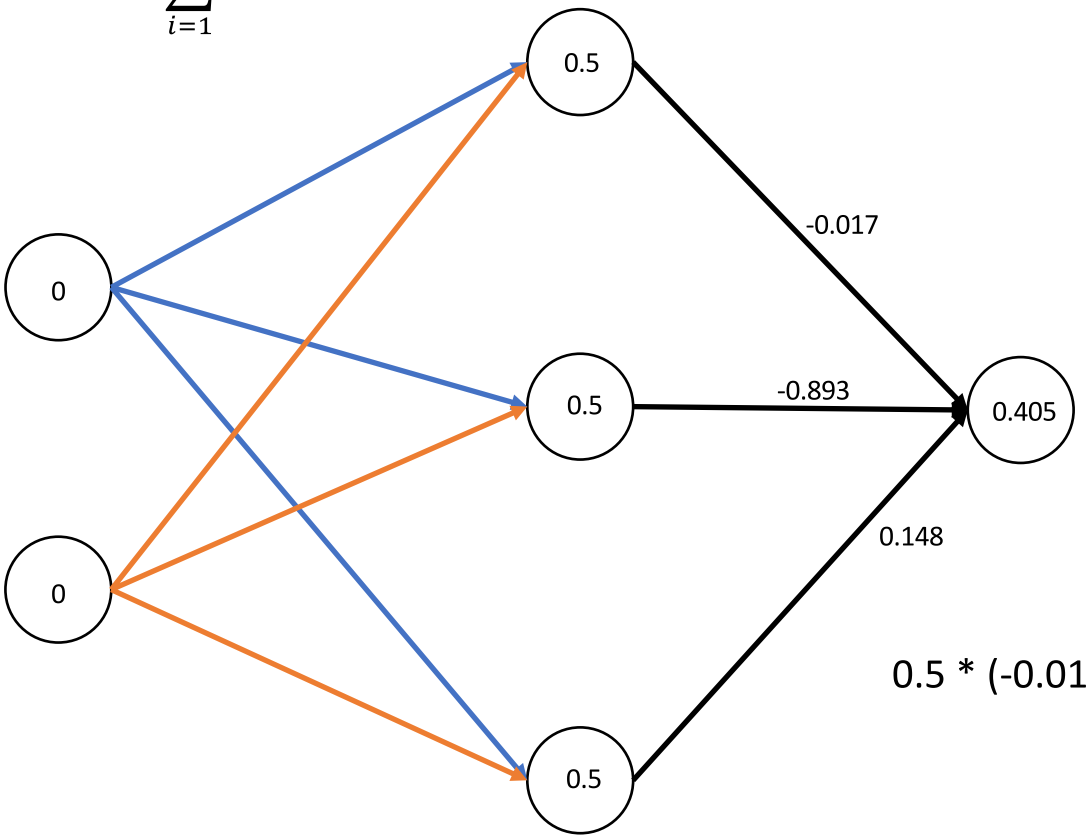
$$1 * (-0.961) + 1 * (-0.469) = -1.430$$

HIDDEN LAYER TO OUTPUT LAYER

$$sum = \sum_{i=1}^{n} xi \; * wi$$

$$y = \frac{1}{1 + e^{-x}}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



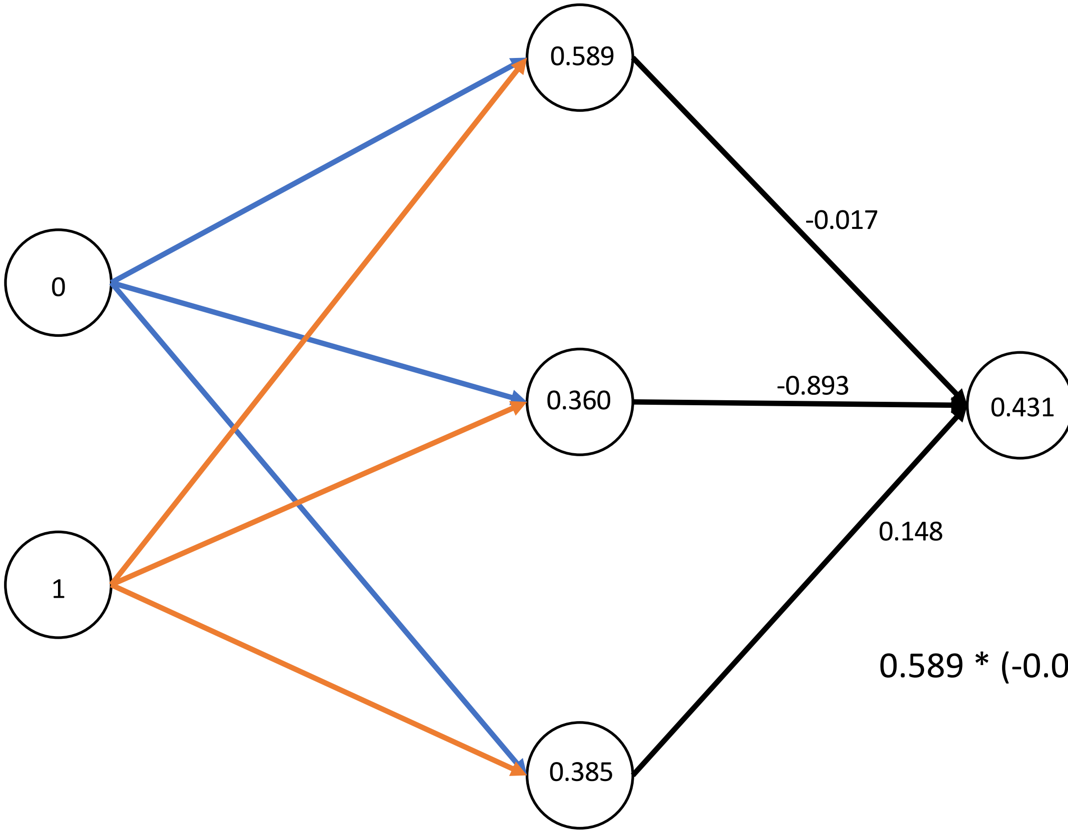
Sum = -0.381

Activation = 0.405

0.5 * (-0.017) + 0.5 * (-0.893) + 0.5 * 0.148 = -0.381

HIDDEN LAYER TO OUTPUT LAYER

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



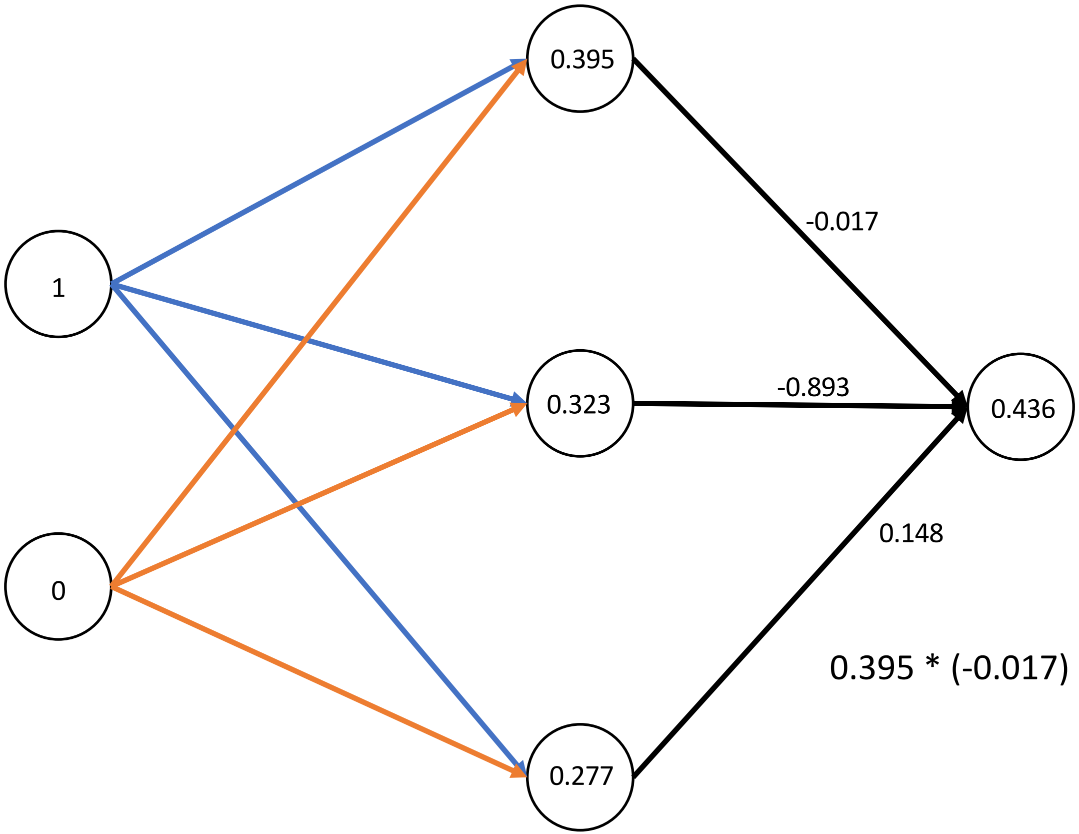
Sum = -0.274

Activation = 0.431

$$0.589 * (-0.017) + 0.360 * (-0.893) + 0.385 * 0.148 = -0.274$$

HIDDEN LAYER TO OUTPUT LAYER

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



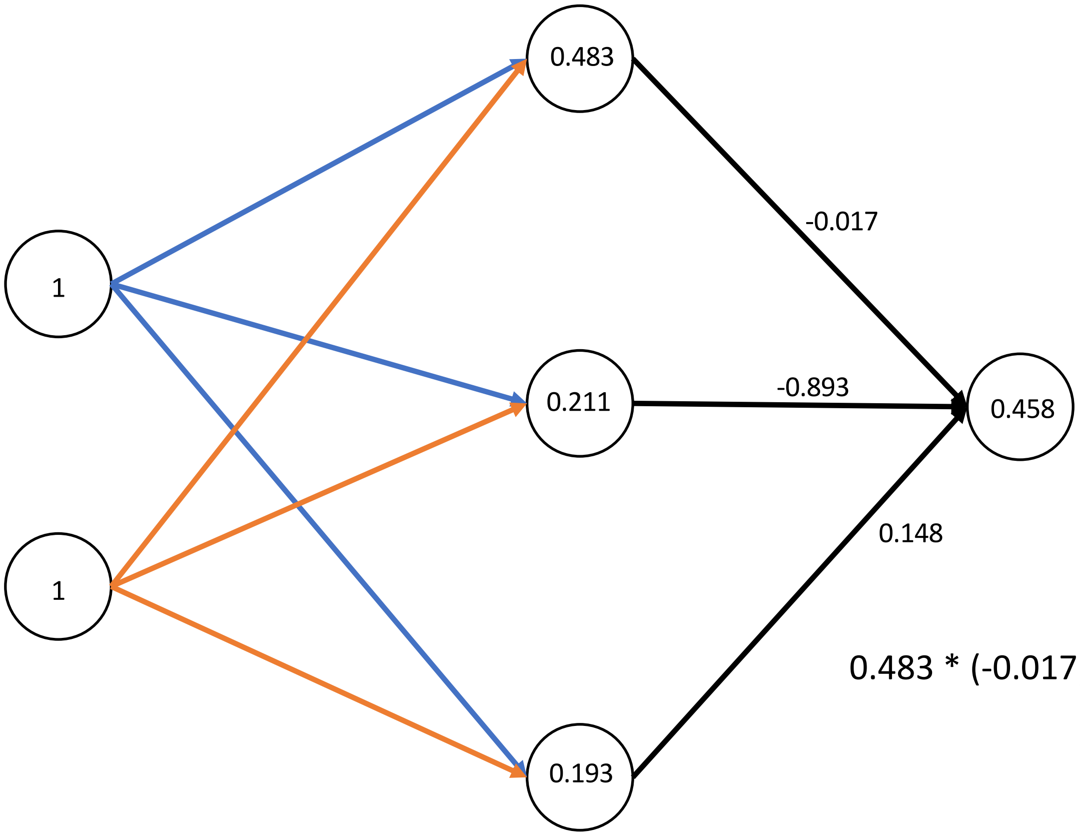
Sum = -0.254

Activation = 0.436

$$0.395 * (-0.017) + 0.323 * (-0.893) + 0.277 * 0.148 = -0.254$$

HIDDEN LAYER TO OUTPUT LAYER

X1	X2	Classe
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.168

Activation = 0.458

0.483 * (-0.017) + 0.211 * (-0.893) + 0.193 * 0.148 = -0.168

“XOR” OPERATOR – ERROR (LOSS FUNCTION)

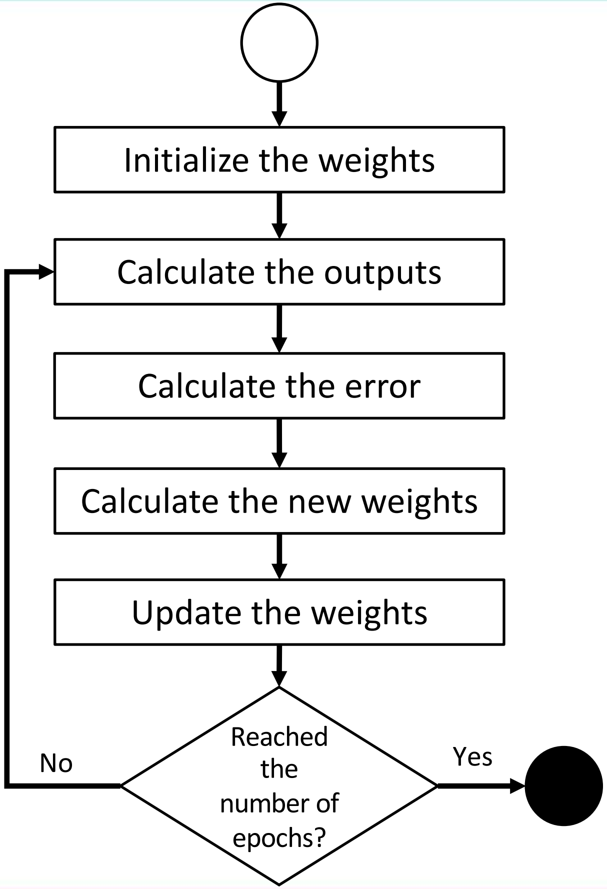
The simplest algorithm

error = correct – prediction

X1	X2	Class	Prediction	Error
0	0	0	0.405	-0.405
0	1	1	0.431	0.569
1	0	1	0.436	0.564
1	1	0	0.458	-0.458

Average = 0.499

ALGORITHM



Cost function (loss function)

Gradient descent

Derivative

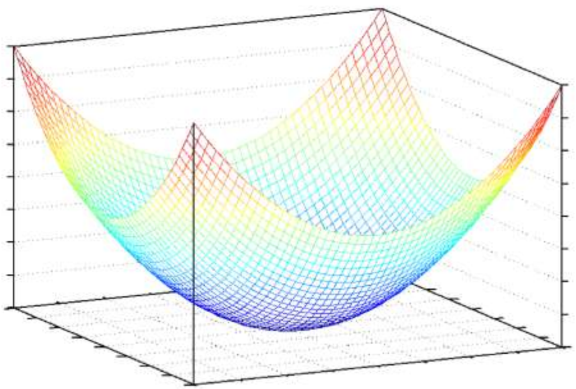
Delta

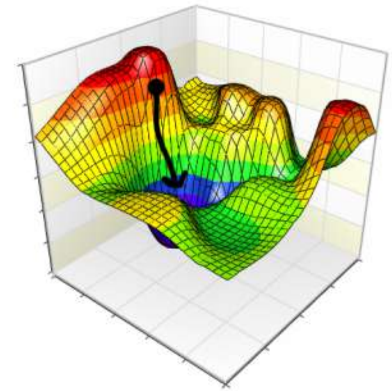
Backpropagation

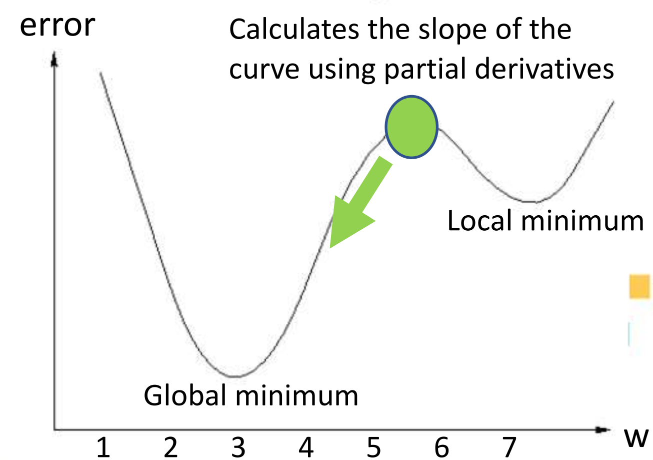
GRADIENT DESCENT

min $C(w_1, w_2 \dots w_n)$

Calculate the partial derivative to move to the gradient direction







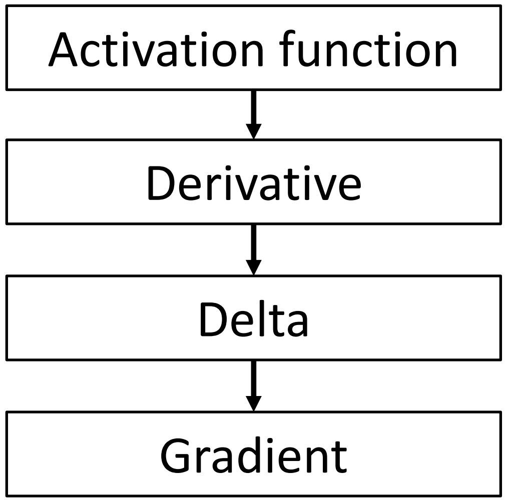
GRADIENT DESCENT (DERIVATIVE)

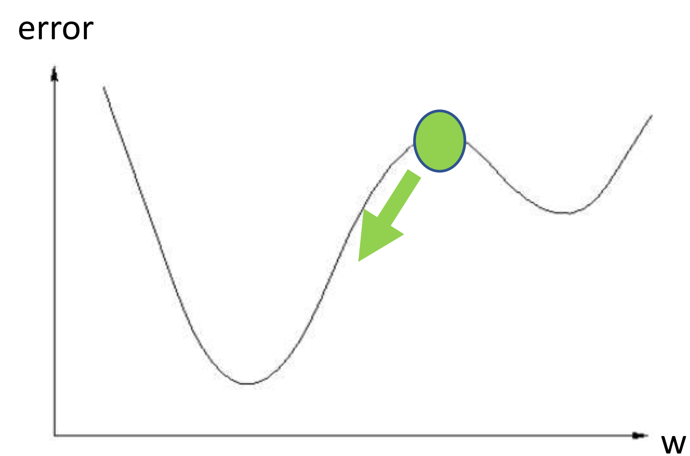
$$y = \frac{1}{1 + e^{-x}}$$

$$d = y * (1 - y)$$

$$d = 0.1 * (1 - 0.1)$$

DELTA PARAMETER

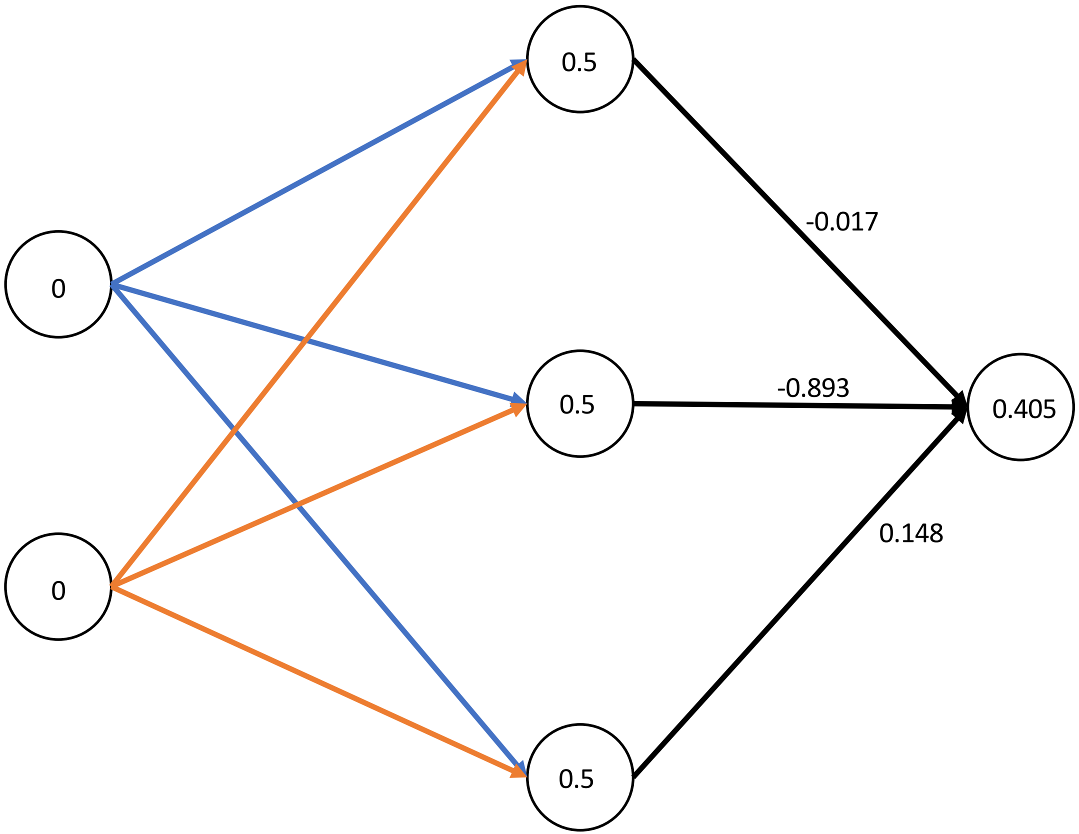




OUTPUT LAYER – DELTA

$$delta_{output} = error * sigmoid_{derivative}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.381

Activation = 0.405

Error = 0 – 0.405 = -0.405

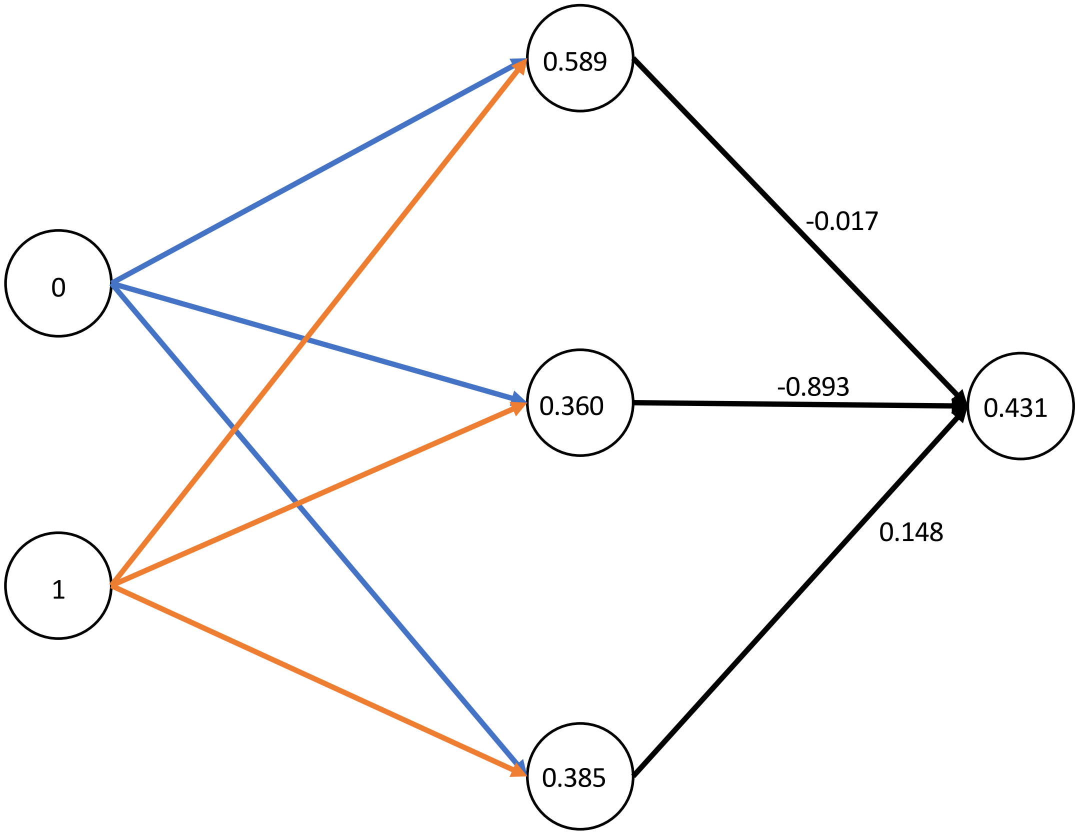
Derivative activation (sigmoid) = 0.241

Delta (output) = -0.405 * 0.241 = -0.097

OUTPUT LAYER – DELTA

$$delta_{output} = error * sigmoid_{derivative}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.274

Activation = 0.431

Error = 1 – 0.431 = 0.569

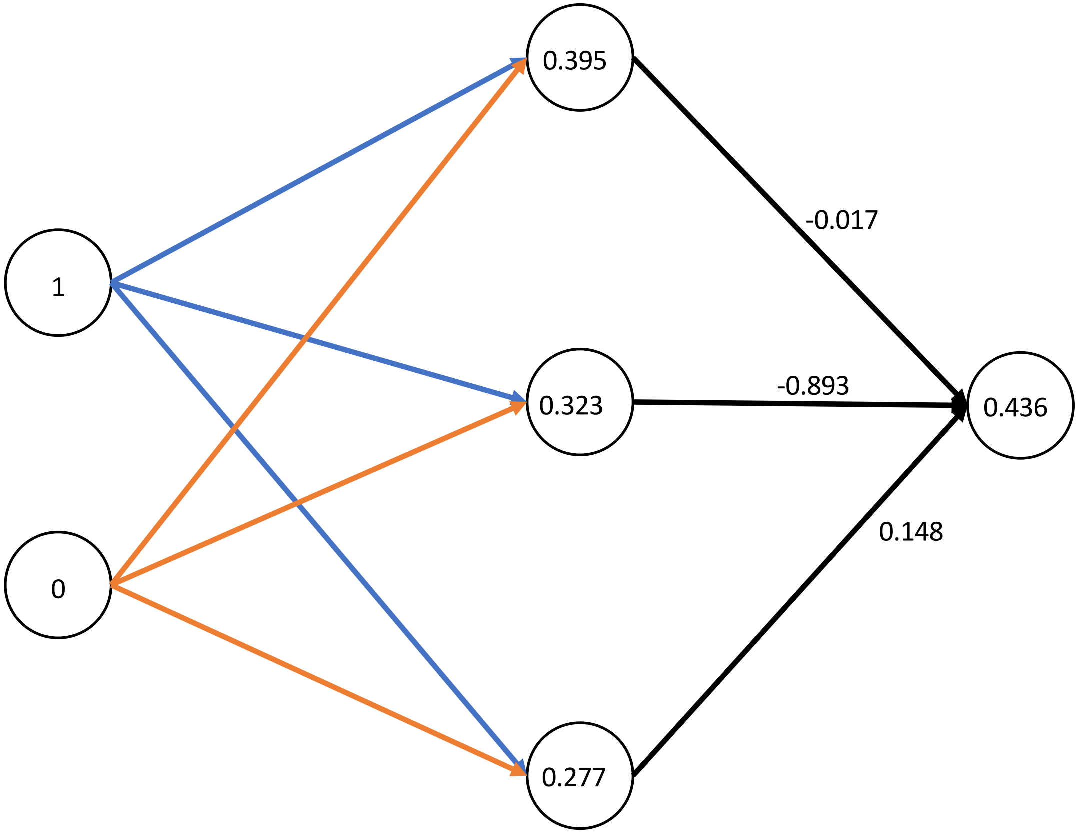
Derivative activation (sigmoid) = 0.245

Delta (output) = 0.569 * 0.245 = 0.139

OUTPUT LAYER – DELTA

$$delta_{output} = error * sigmoid_{derivative}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.254

Activation = 0.436

Error = 1 – 0.436 = 0.564

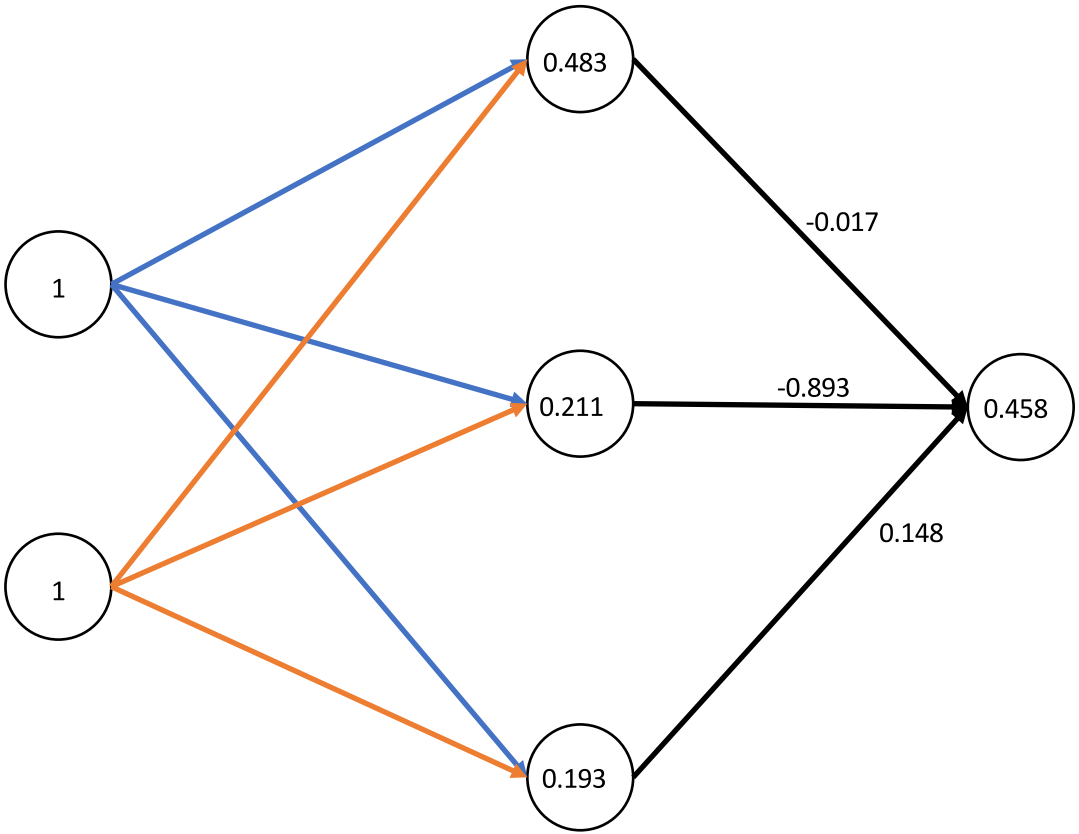
Derivative activation (sigmoid) = 0.246

Delta (output) = 0.564 * 0.246 = 0.138

OUTPUT LAYER – DELTA

$$delta_{output} = error * sigmoid_{derivative}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.168

Activation = 0.458

Error = 0 – 0.458 = -0.458

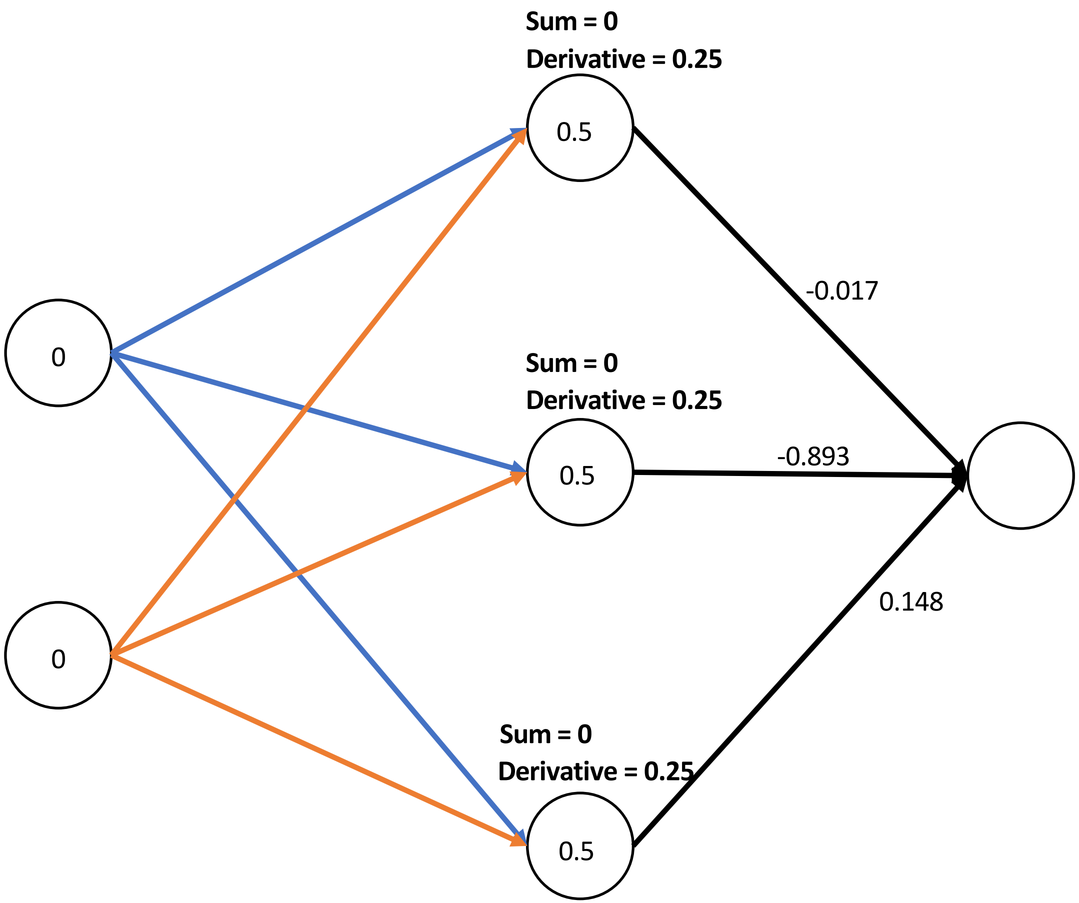
Derivative activation (sigmoid) = 0.248

Delta (output) = -0.458 * 0.248 = -0.113

HIDDEN LAYER – DELTA

$$delta_{hidden} = sigmoid_{derivative} * weight * delta_{output}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Delta (output) = -0.097

0.25 * (-0.017) * (-0.097) = 0.000

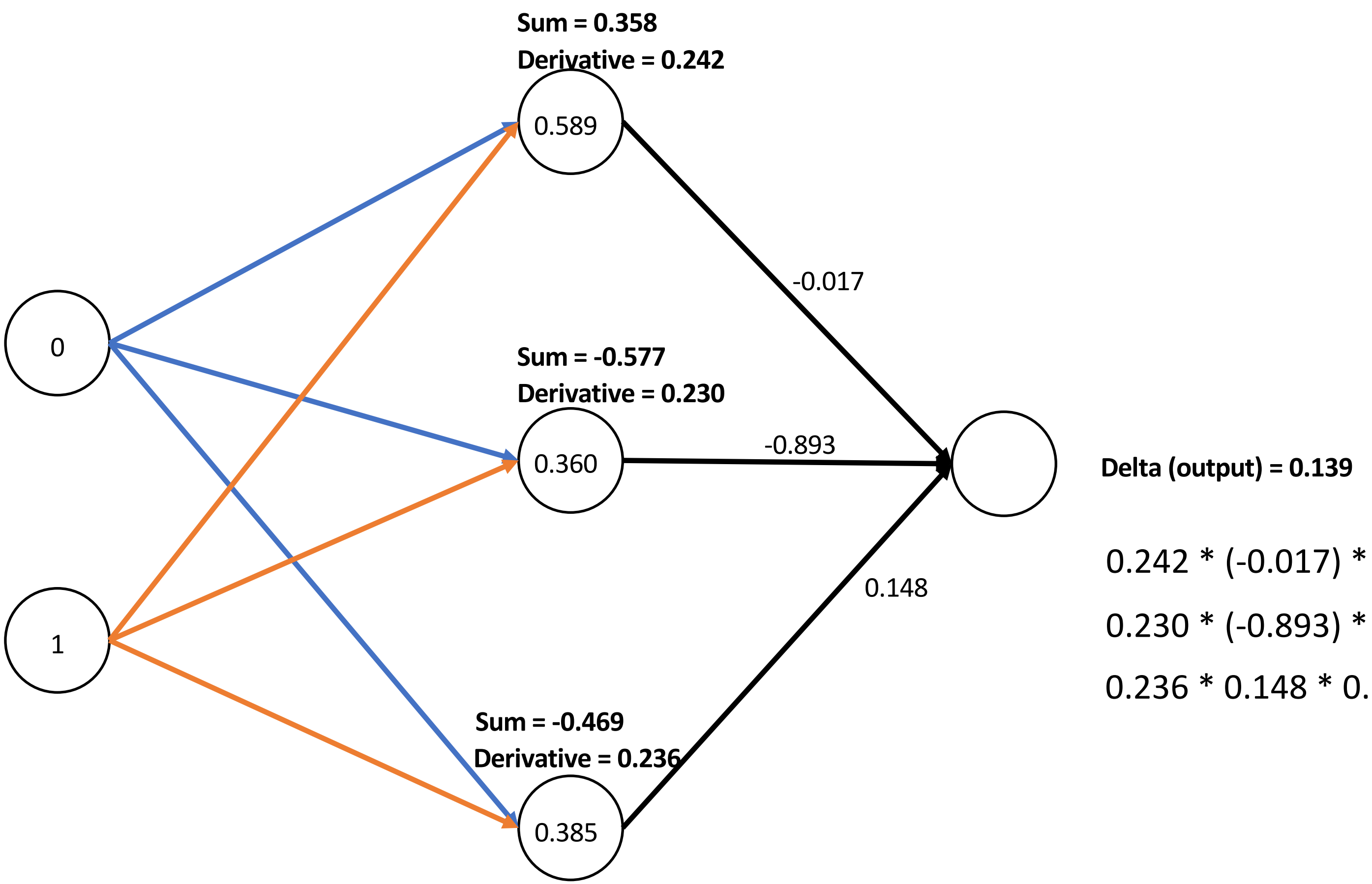
0.25 * (-0.893) * (-0.097) = 0.021

0.25 * 0.148 * (-0.097) = -0.003

HIDDEN LAYER – DELTA

$$delta_{hidden} = sigmoid_{derivative} * weight * delta_{output}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



0.242 * (-0.017) * 0.139 = -0.000

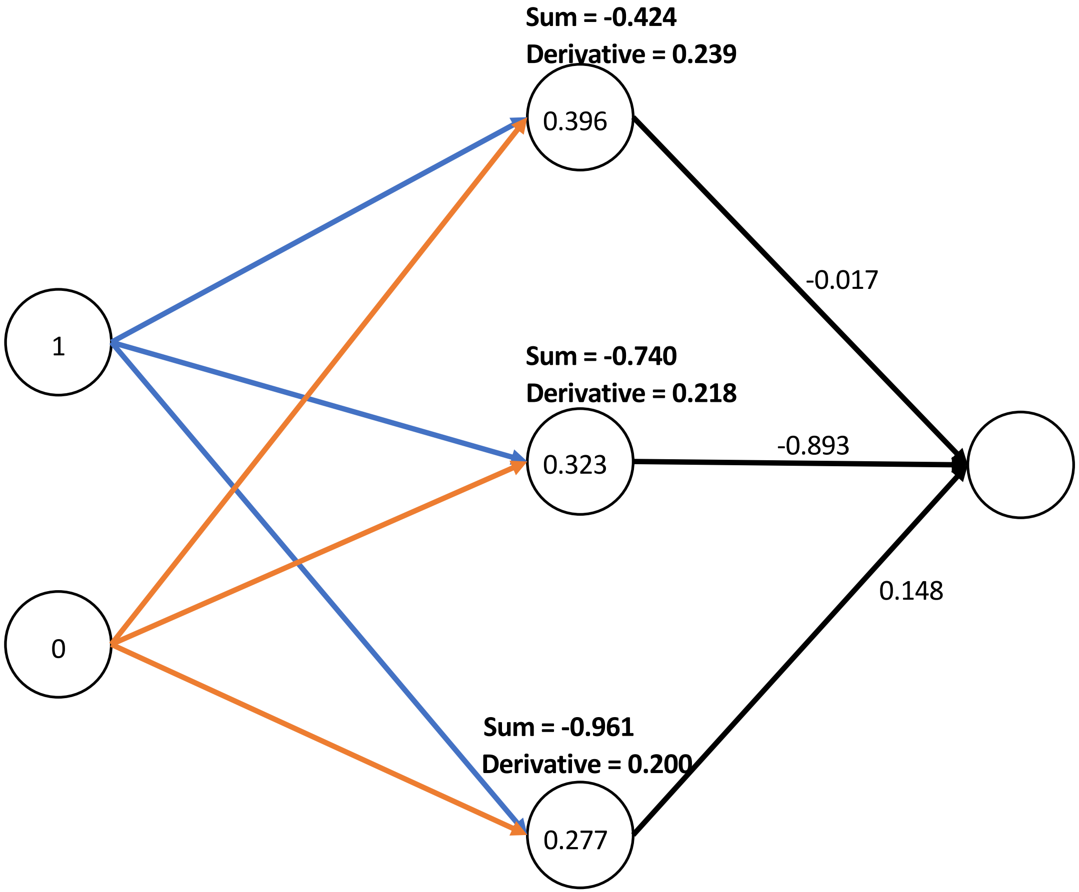
0.230 * (-0.893) * 0.139 = -0.028

0.236 * 0.148 * 0.139 = 0.004

HIDDEN LAYER – DELTA

$$delta_{hidden} = sigmoid_{derivative} * weight * delta_{output}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Delta (output) = 0.138

0.239 * (-0.017) * 0.138 = -0.000

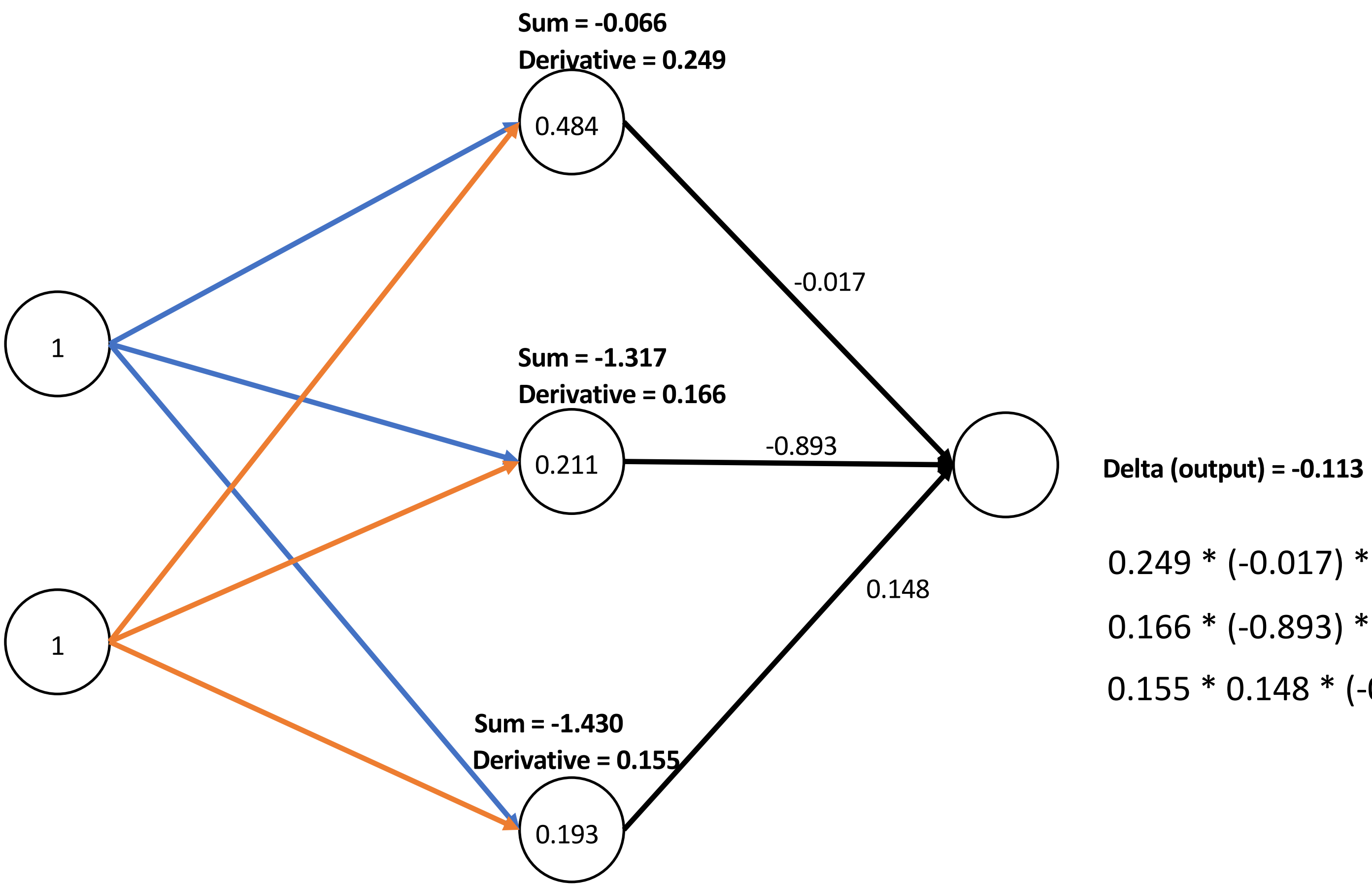
0.218 * (-0.893) * 0.138 = -0.026

0.200 * 0.148 * 0.138 = 0.004

HIDDEN LAYER – DELTA

$$delta_{hidden} = sigmoid_{derivative} * weight * delta_{output}$$

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

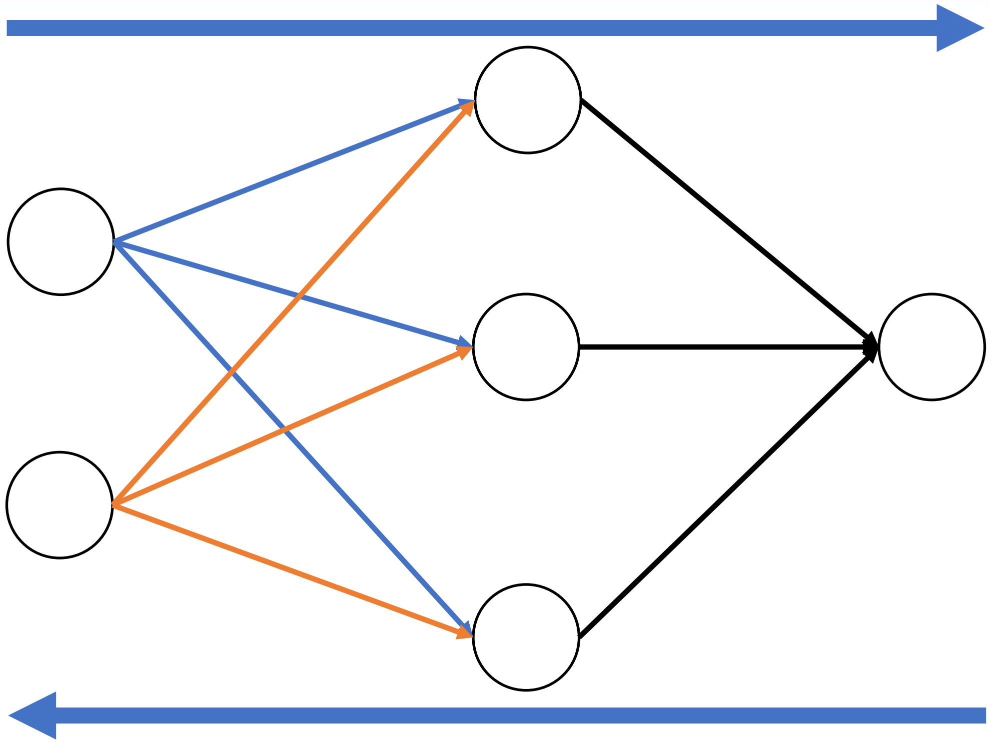


0.249 * (-0.017) * (-0.113) = 0.000

0.166 * (-0.893) * (-0.113) = 0.016

0.155 * 0.148 * (-0.113) = -0.002

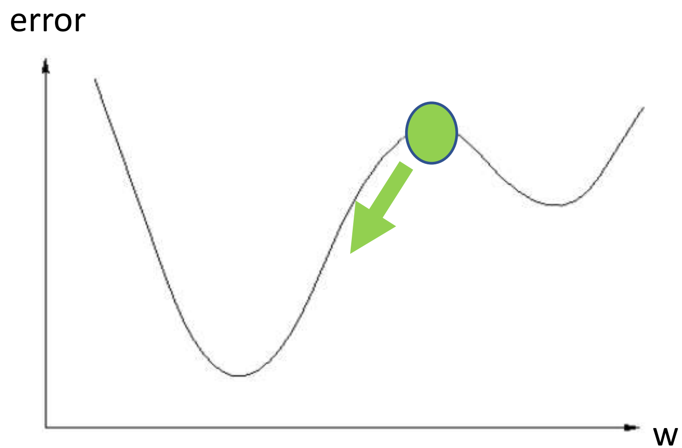
WEIGHT UPDATE



$$weight_{n+1} = weight_n + (input * delta * learing\_rate)$$

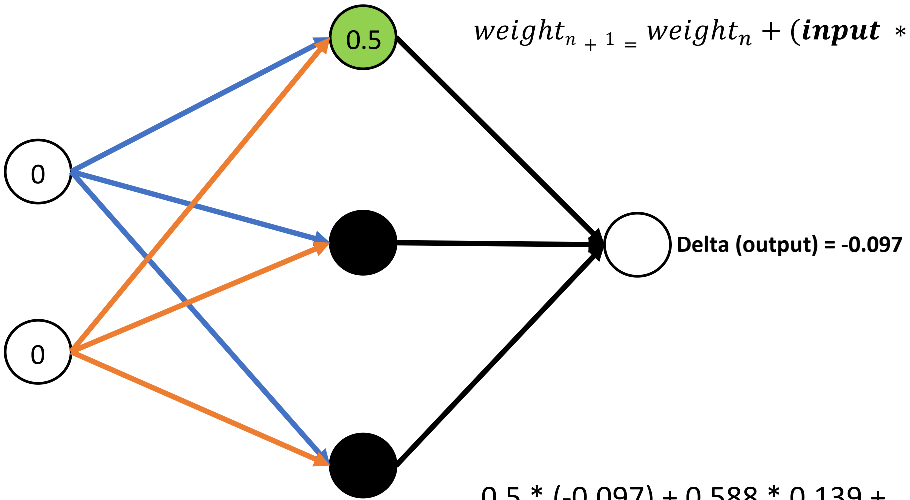
LEARNING RATE

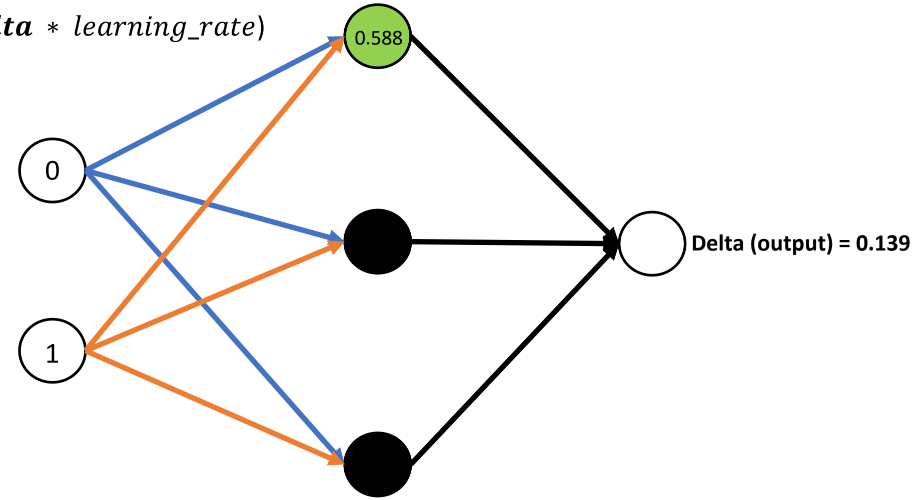
- Defines how fast the algorithm will learn
- High: the convergence is fast but may lose the global minimum
- Low: the convergence will be slower but more likely to reach the global minimum



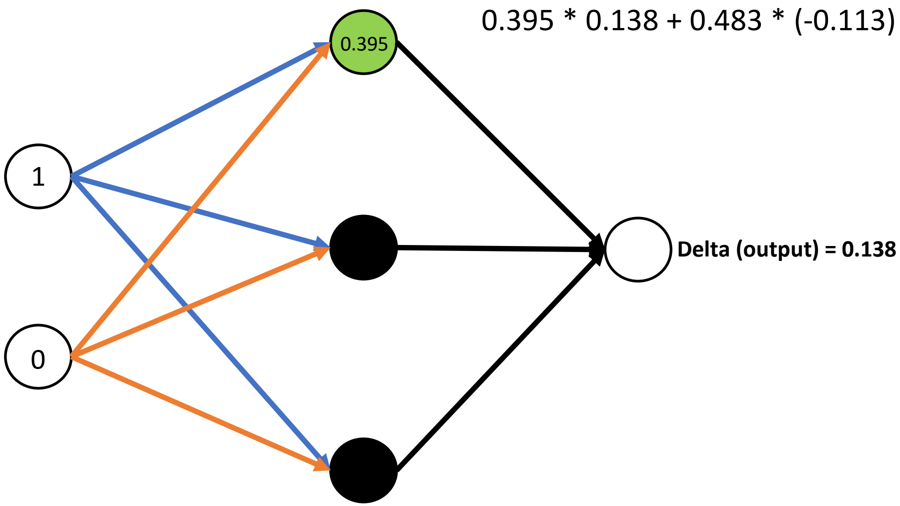
WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER

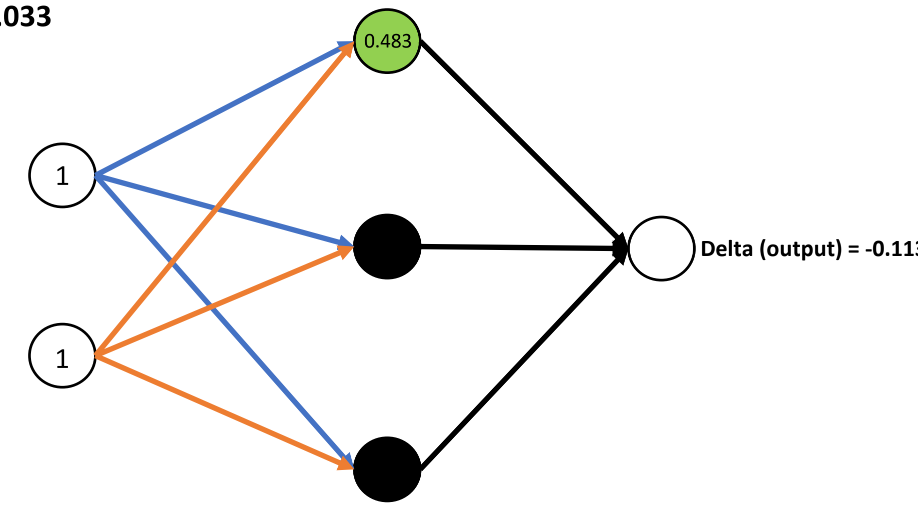
$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$





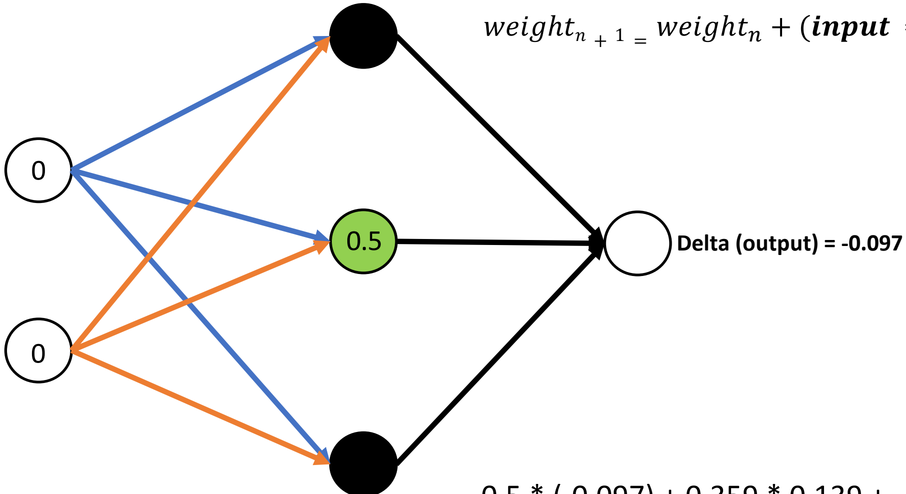
0.5 * (-0.097) + 0.588 * 0.139 +
0.395 * 0.138 + 0.483 * (-0.113) = **0.033**

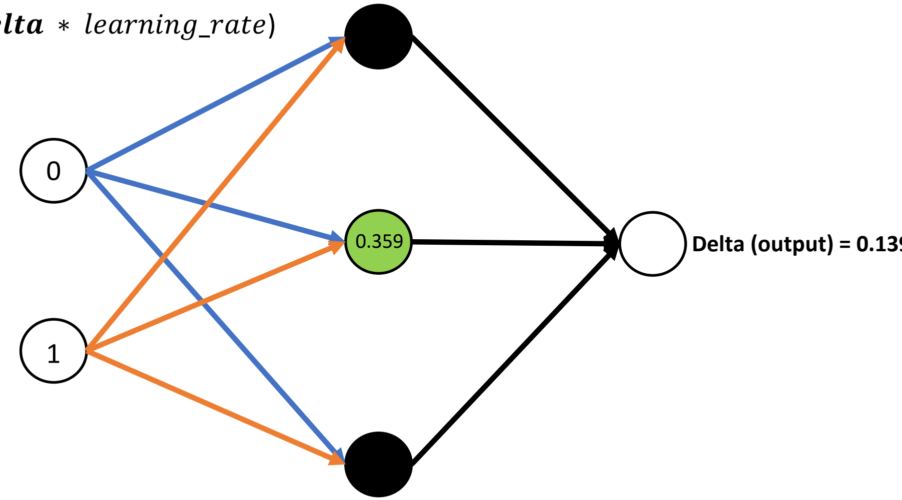




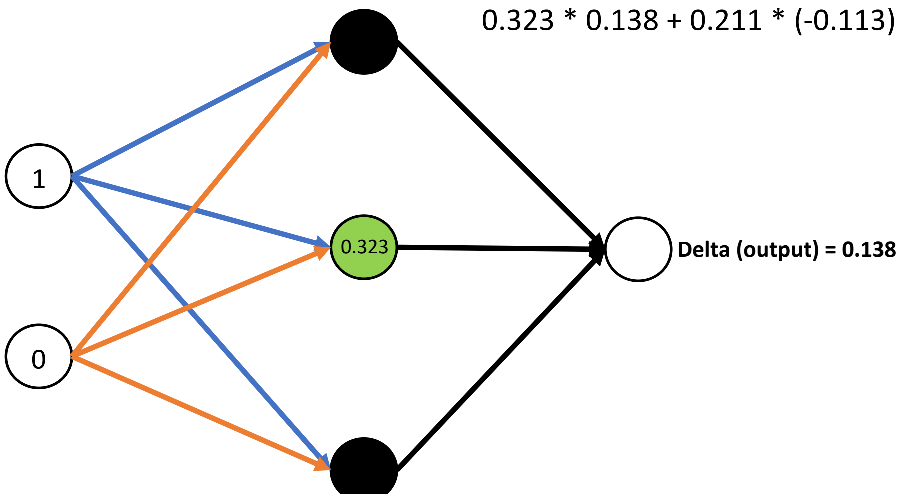
WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER

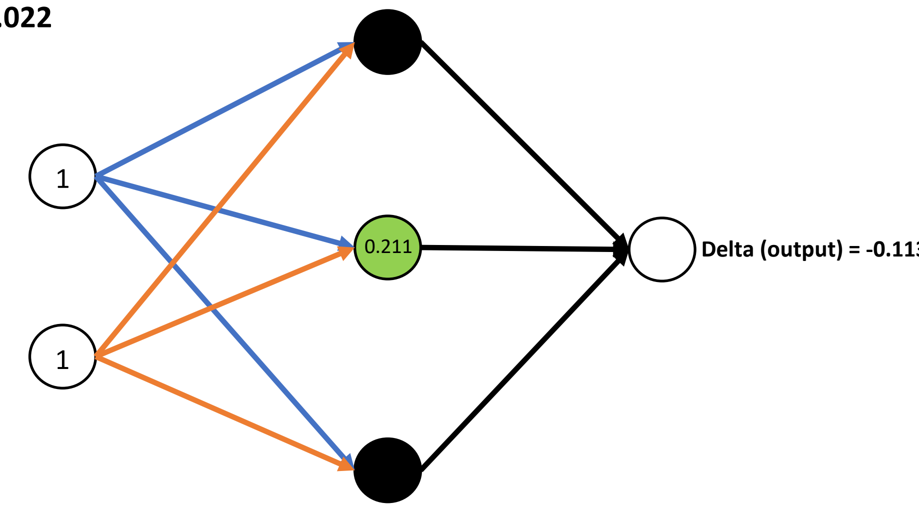
$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$





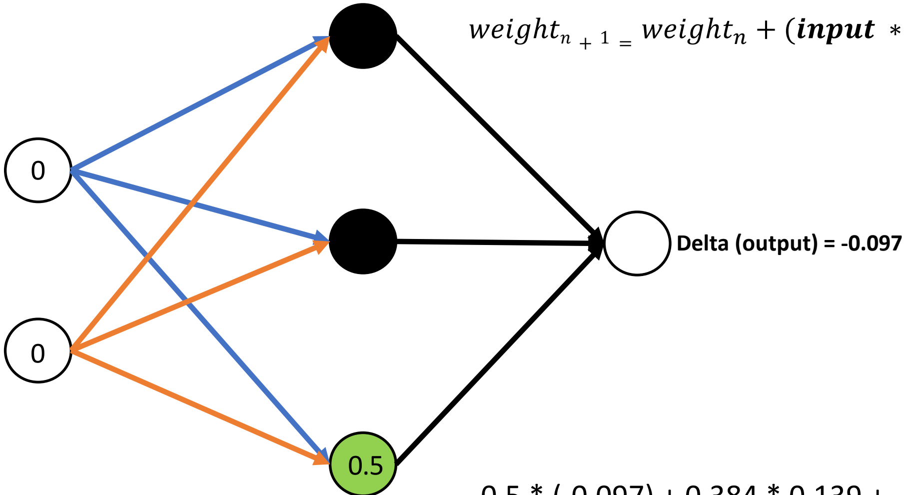
0.5 * (-0.097) + 0.359 * 0.139 +
0.323 * 0.138 + 0.211 * (-0.113) = **0.022**

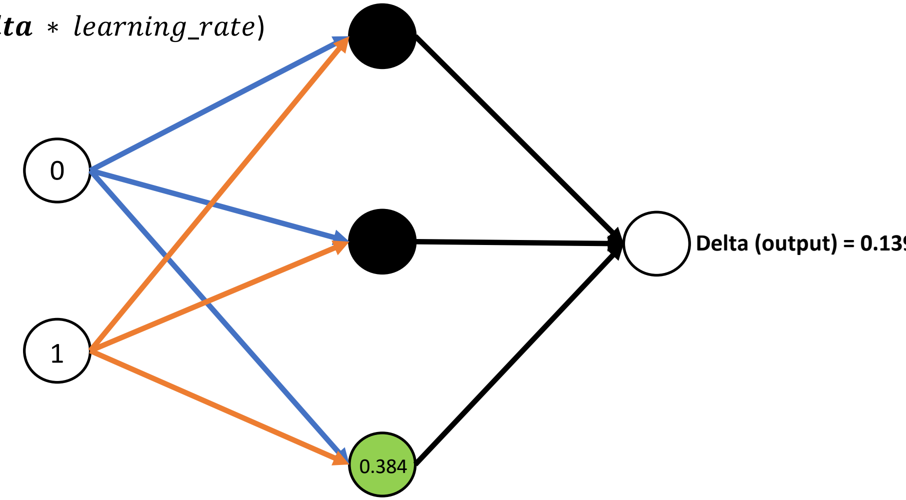




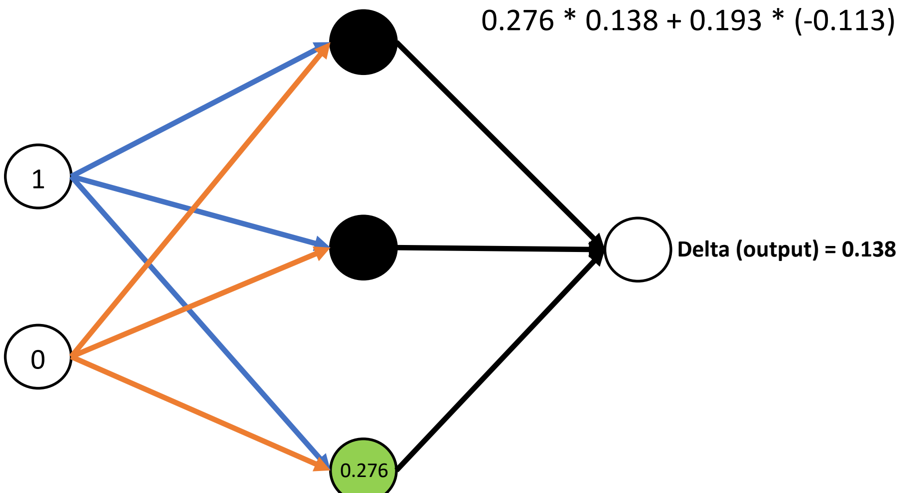
WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER

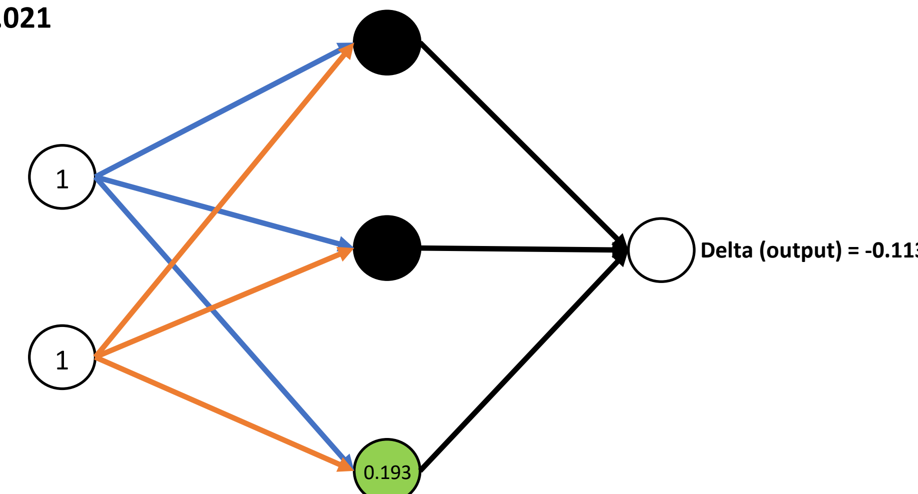
$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$





0.5 * (-0.097) + 0.384 * 0.139 +
0.276 * 0.138 + 0.193 * (-0.113) = **0.021**

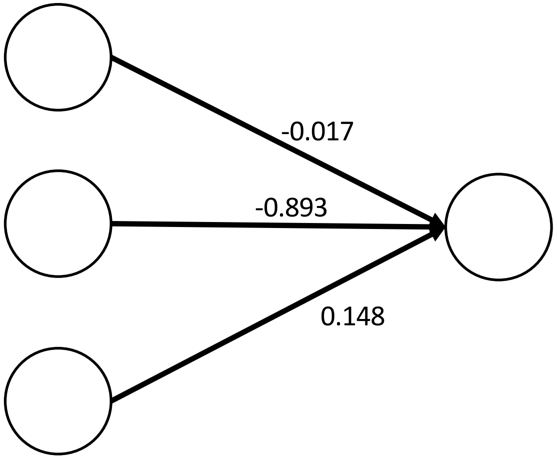




WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER

Learning rate = 0.3

Input x delta
0.033
0.022
0.021

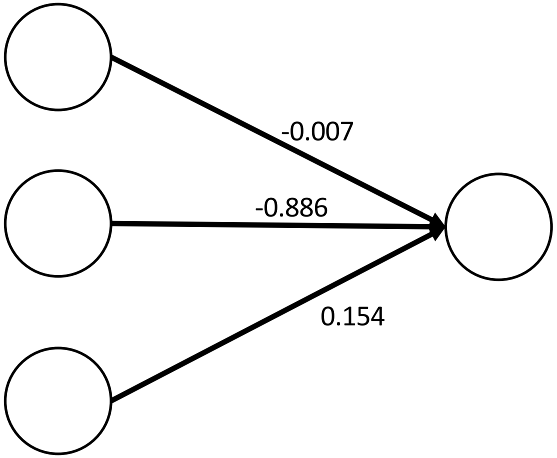


$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$

-0.017 + 0.033 * 0.3 = **-0.007**

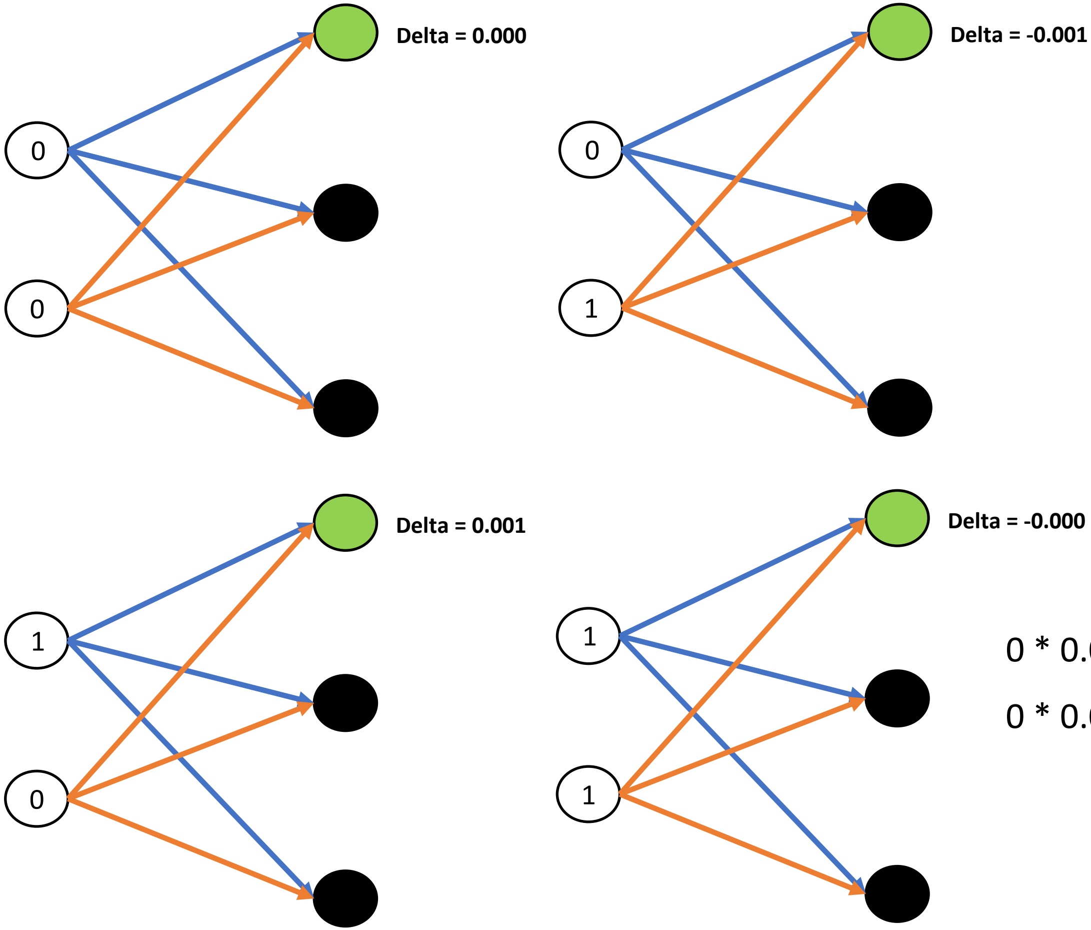
-0.893 + 0.022 * 0.3 = **-0.886**

0.148 + 0.021 * 0.3 = **0.154**



WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER

$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$

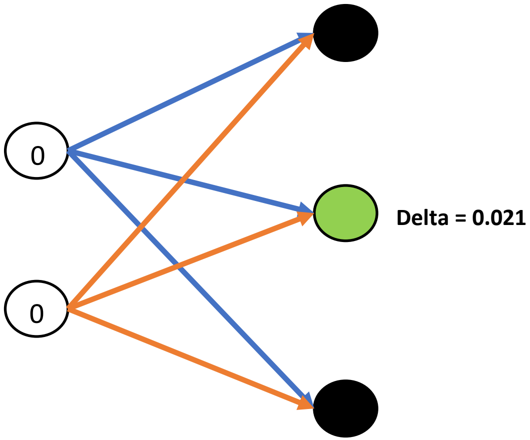


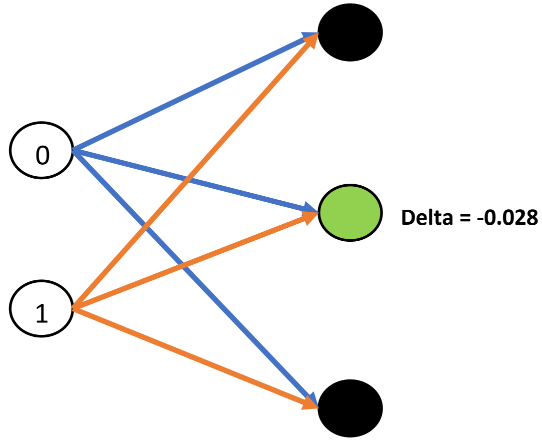
0 * 0.000 + 0 * (-0.001) + 1 * (0.001) + 1 * -0.000 = -0.000

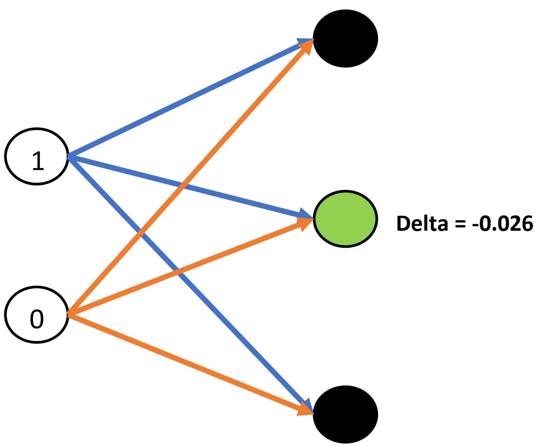
0 * 0.000 + 1 * (-0.001) + 0 * (0.001) + 1 * -0.000 = -0.000

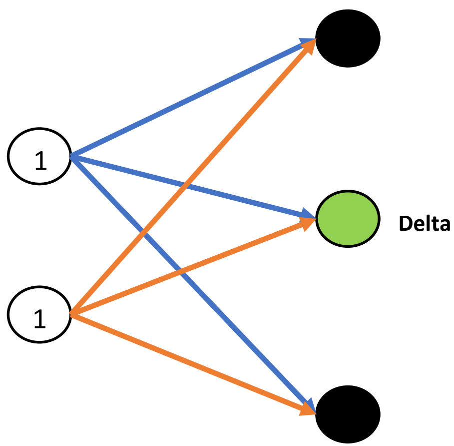
WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER

$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$









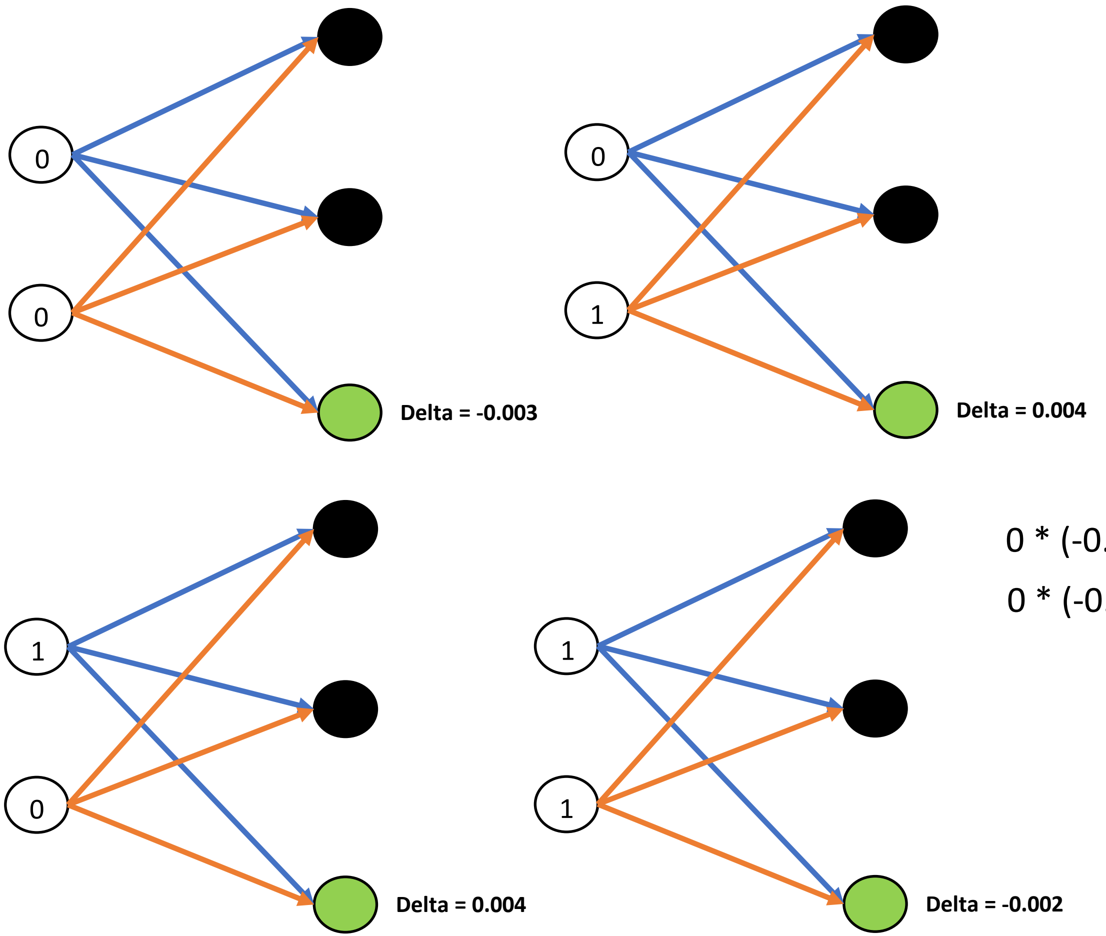
Delta = 0.016

0 * 0.021 + 0 * (-0.028) + 1 * (-0.026) + 1 * 0.016 = -0.009

0 * 0.021 + 1 * (-0.028) + 0 * (-0.026) + 1 * 0.016 = -0.012

WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER

$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$



0 * (-0.003) + 0 * 0.004 + 1 * 0.004 + 1 * (-0.002) = 0.002

0 * (-0.003) + 1 * 0.004 + 0 * 0.004 + 1 * (-0.002) = 0.002

WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER

Learning rate = 0.3

Input x delta

-0.000 -0.009 0.002

-0.000 -0.012 0.002

$$weight_{n+1} = weight_n + (\boldsymbol{input} * \boldsymbol{delta} * learning\_rate)$$

-0.424 + (-0.000) * 0.3 = **-0.424**

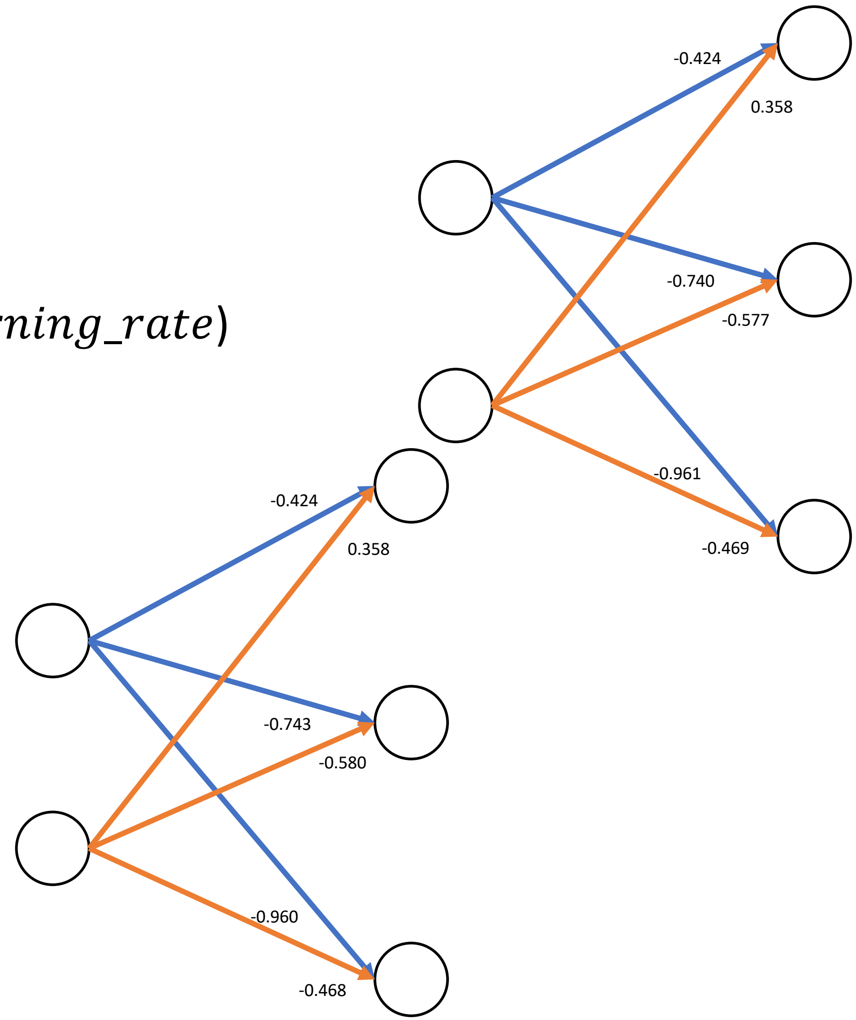
0.358 + (-0.000) * 0.3 = **0.358**

-0.740 + (-0.009) * 0.3 = **-0.743**

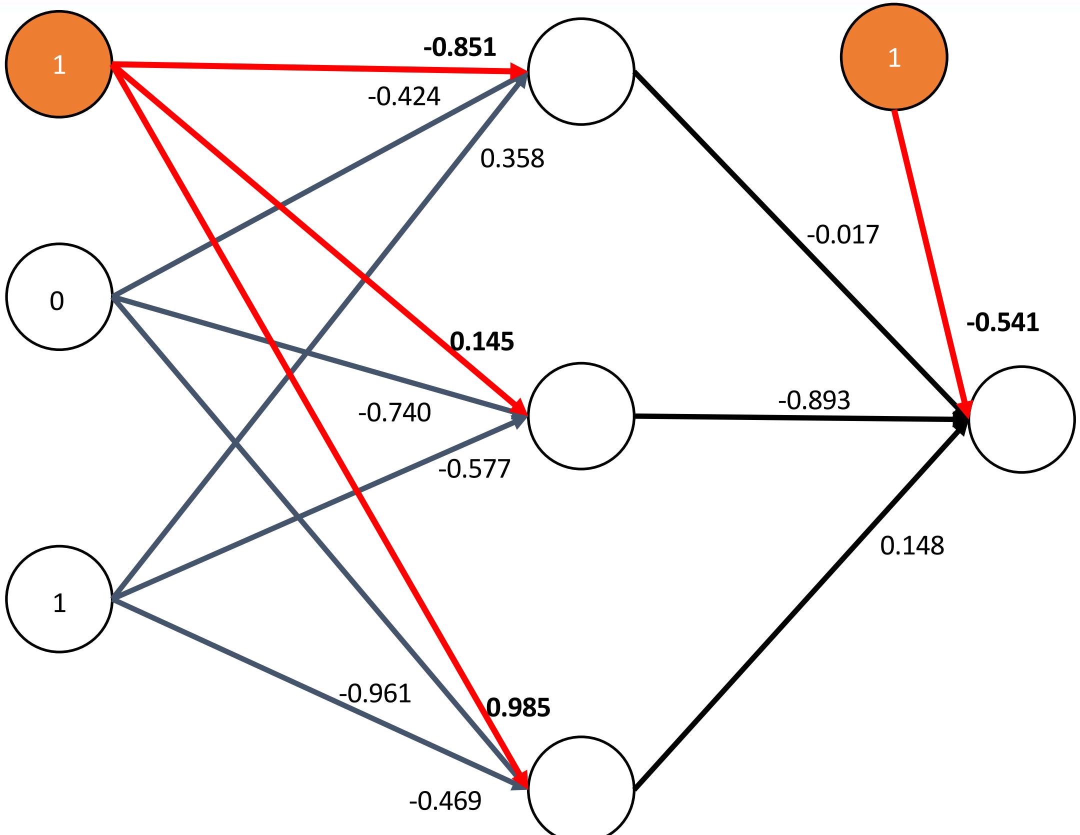
-0.577 + (-0.012) * 0.3 = **-0.580**

-0.961 + 0.002 * 0.3 = **-0.960**

-0.469 + 0.002 * 0.3 = **-0.468**



BIAS



$$output = sum(inputs * weights) + bias$$

ERROR (LOSS FUNCTION)

The simplest algorithm

error = correct – prediction

X1	X2	Class	Prediction	Error
0	0	0	0.405	-0.405
0	1	1	0.431	0.569
1	0	1	0.436	0.564
1	1	0	0.458	-0.458

Average = 0.499

python

MEAN SQUARED ERROR (MSE) AND ROOT MEAN SQUARED ERROR (RMSE)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

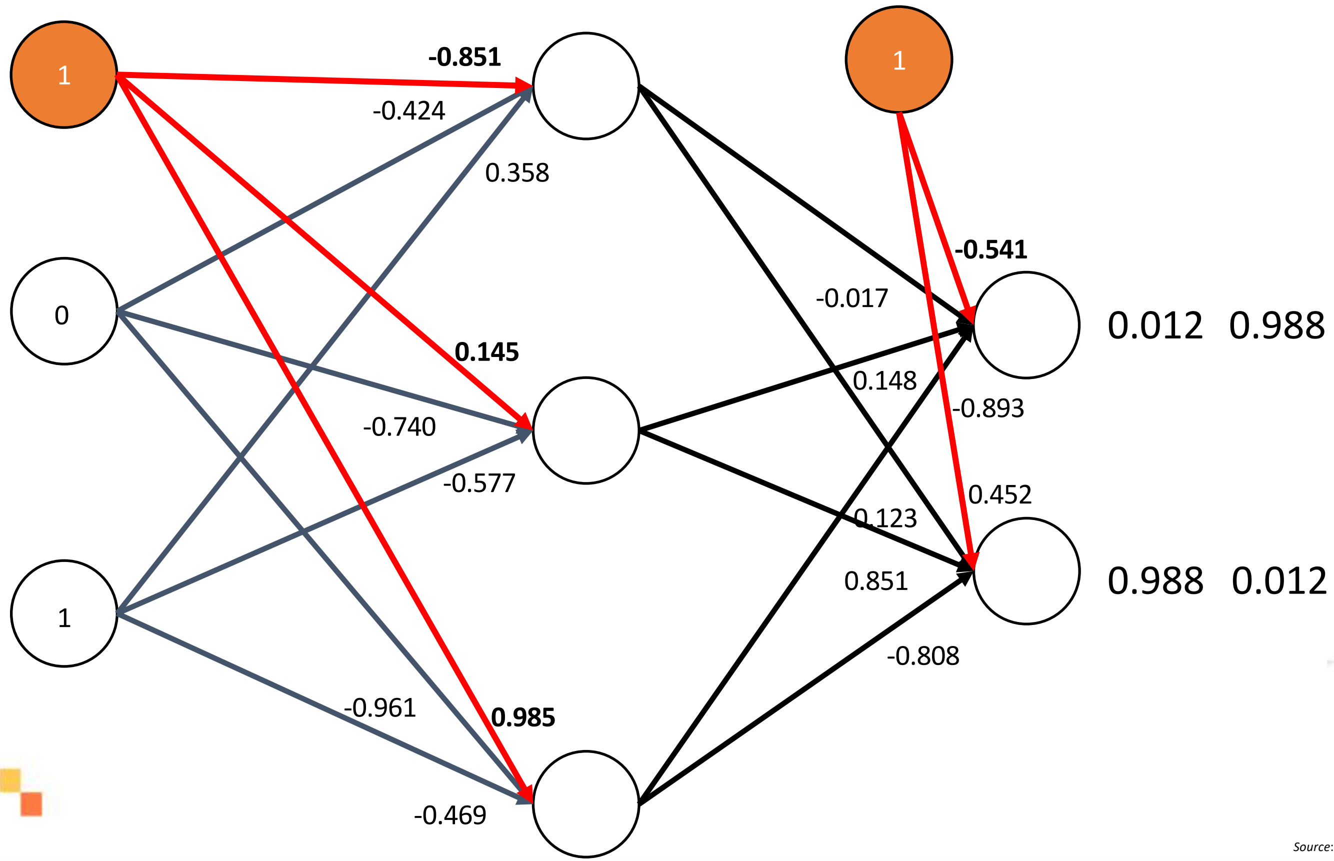
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - o_i)^2}$$

X1	X2	Class	Prediction	Error
0	0	0	0.405	$(0 - 0.405)^2 = 0.164$
0	1	1	0.431	$(1 - 0.431)^2 = 0.322$
1	0	1	0.436	$(1 - 0.436)^2 = 0.316$
1	1	0	0.458	$(0 - 0.458)^2 = 0.209$

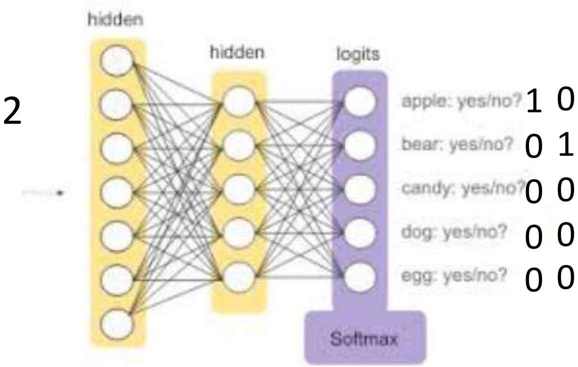
10 (expected) – 5 (prediction) = 5 ($5^2 = 25$)
10 (expected) – 8 (prediction) = 2 ($2^2 = 4$)

Sum = 1.011
MSE = 1.011 / 4 = 0.252
RMSE = 0.501

MULTIPLE OUTPUTS



X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

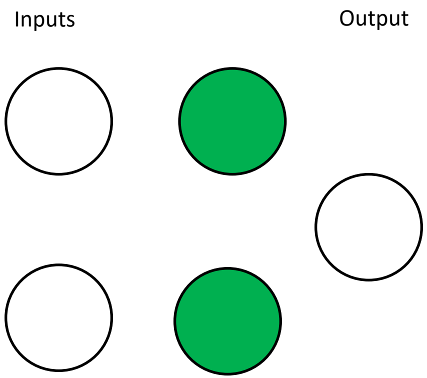


Source: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

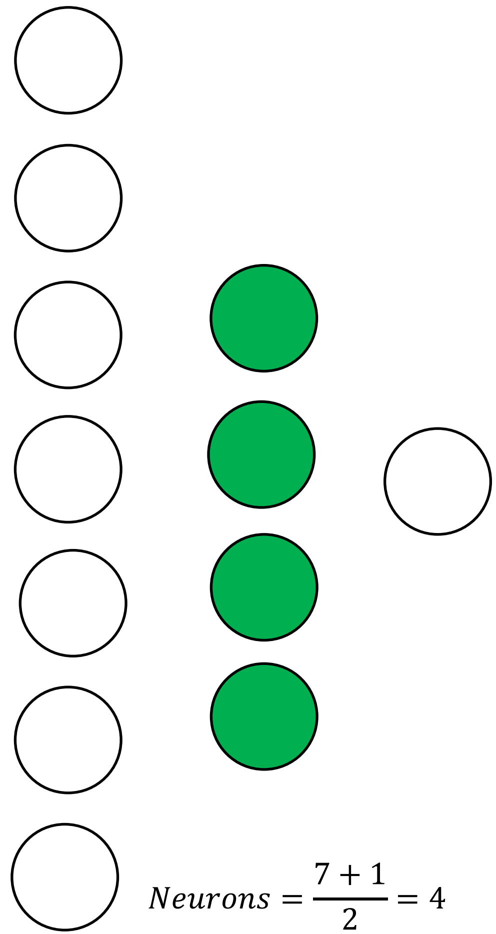
python

HIDDEN LAYERS

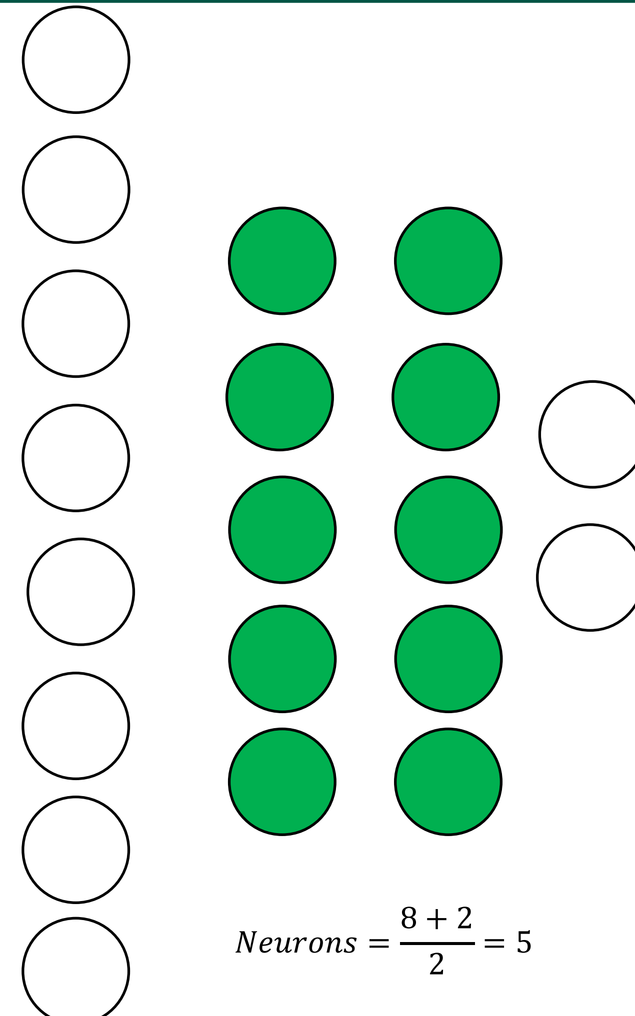
$$Neurons = \frac{Inputs + Outputs}{2}$$



$$Neurons = \frac{2+1}{2} = 1.5$$



$$Neurons = \frac{7+1}{2} = 4$$



$$Neurons = \frac{8+2}{2} = 5$$

HIDDEN LAYERS

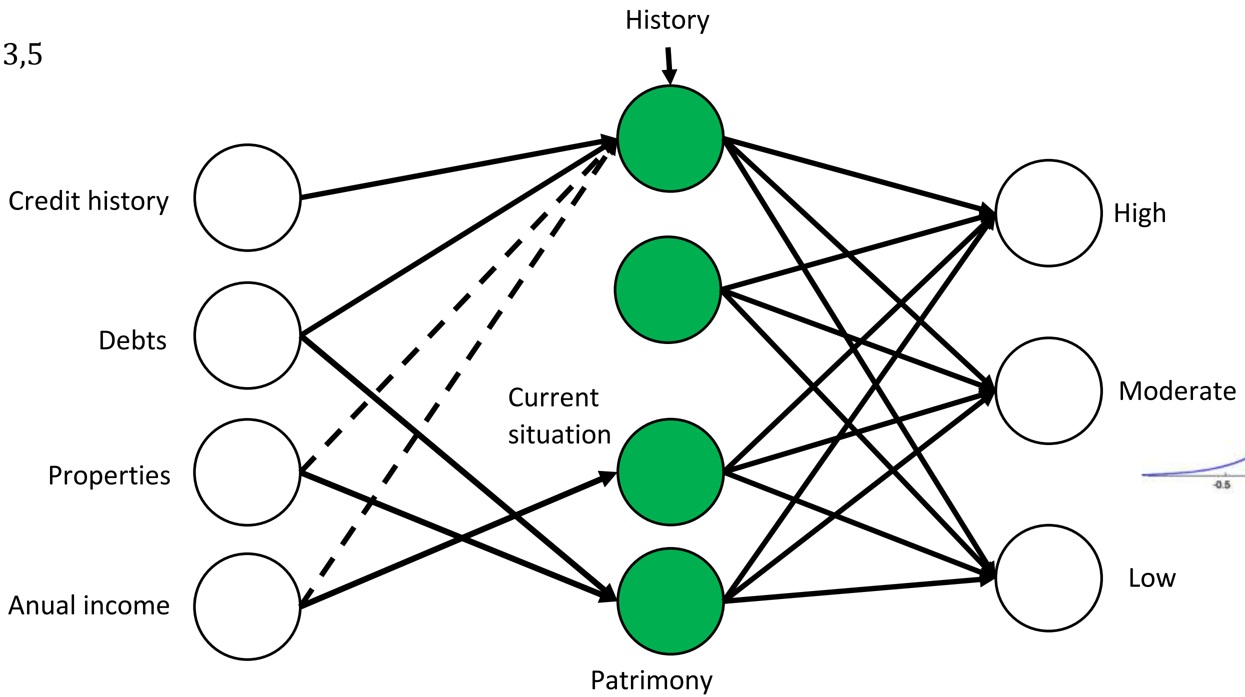
- Linearly separable problems do not require hidden layers
- In general, two layers work well
- Deep learning research shows that more layers are essential for more complex problems

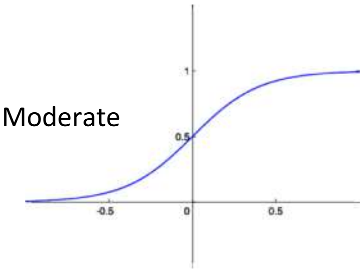
HIDDEN LAYERS

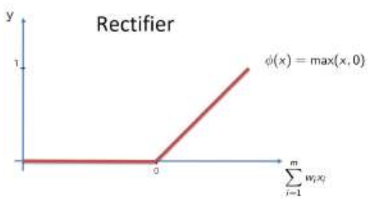
Credit history	Debts	Properties	Anual income	Risk
Bad	High	No	< 15.000	High
Unknown	High	No	>= 15.000 a <= 35.000	High
Unknown	Low	No	>= 15.000 a <= 35.000	Moderate
Unknown	Low	No	> 35.000	High
Unknown	Low	No	> 35.000	Low
Unknown	Low	Yes	> 35.000	Low
Bad	Low	No	< 15.000	High
Bad	Low	Yes	> 35.000	Moderate
Good	Low	No	> 35.000	Low
Good	High	Yes	> 35.000	Low
Good	High	No	< 15.000	High
Good	High	No	>= 15.000 a <= 35.000	Moderate
Good	High	No	> 35.0000	Low
Bad	High	No	>= 15.000 a <= 35.000	High

HIDDEN LAYERS

$$Neurons = \frac{4+3}{2} = 3{,}5$$

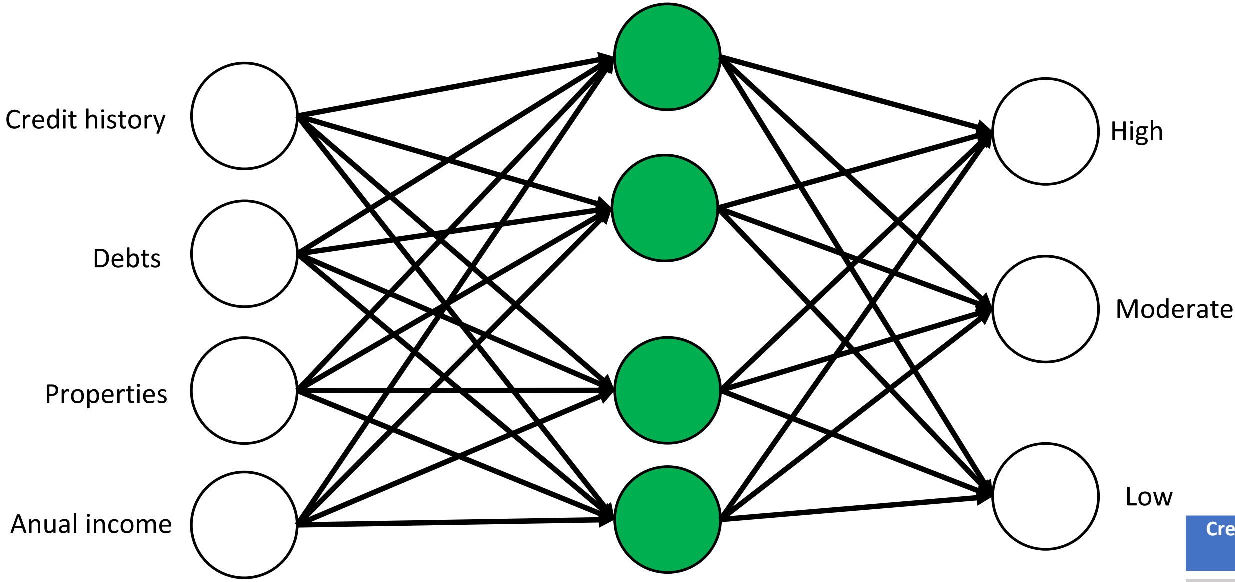






The higher the activation value, the more impact the neuron has

OUTPUT LAYER WITH CATEGORICAL DATA



error = correct – prediction

expected output = 1 0 0

prediction = 0.95 0.02 0.03

error = (1 – 0.95) + (0 – 0.02) + (0 – 0.03)

error = 0.05 + 0.02 + 0.03 = 0.08

Credit history	Debts	Properties	Anual income	Risk
3	1	1	1	1 0 0
2	1	1	2	1 0 0
2	2	1	2	0 1 0
2	2	1	3	1 0 0
2	2	1	3	0 0 1
2	2	2	3	0 0 1
3	2	1	1	1 0 0
3	2	2	3	0 1 0
1	2	1	3	0 0 1
1	1	2	3	0 0 1
1	1	1	1	1 0 0
1	1	1	2	0 1 0
1	1	1	3	0 0 1
3	1	1	2	1 0 0

STOCHASTIC GRADIENT DESCENT

Credit history	Debts	Properties	Anual income	Risk
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

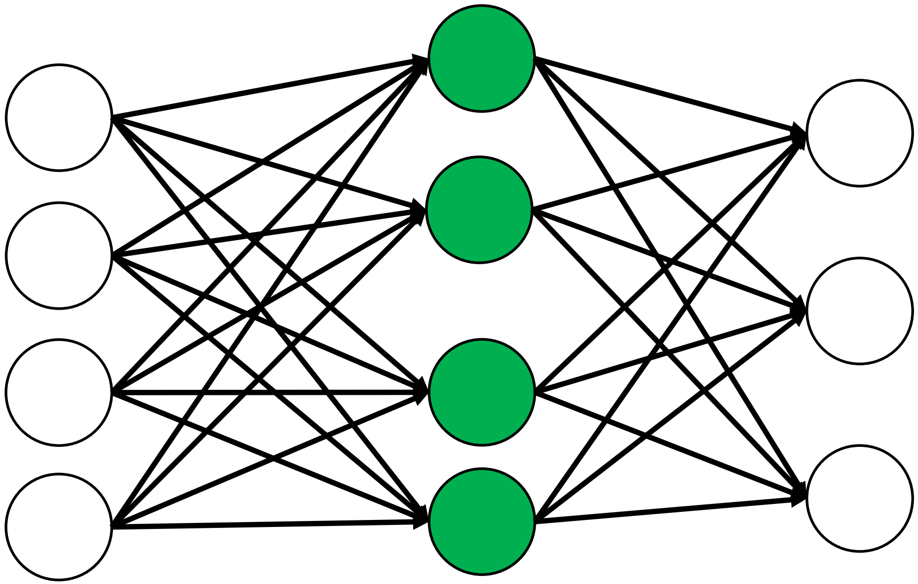
Credit history	Debts	Properties	Anual income	Risk
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

Batch gradient descent

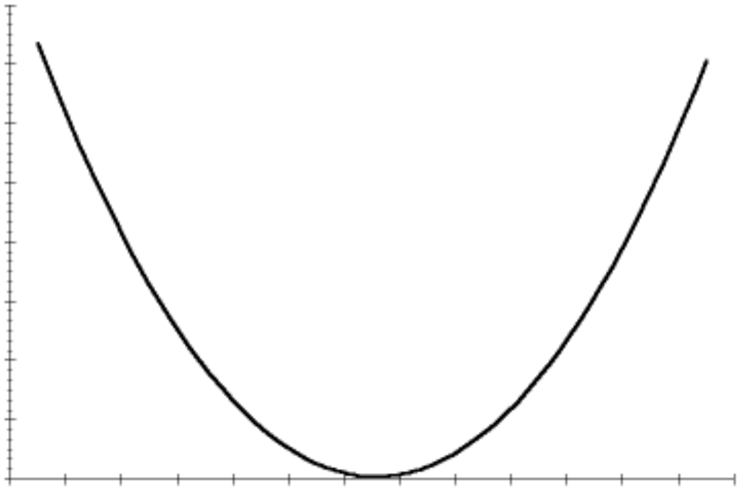
Calculate the error for all instances and then update the weights

Stochastic gradient descent

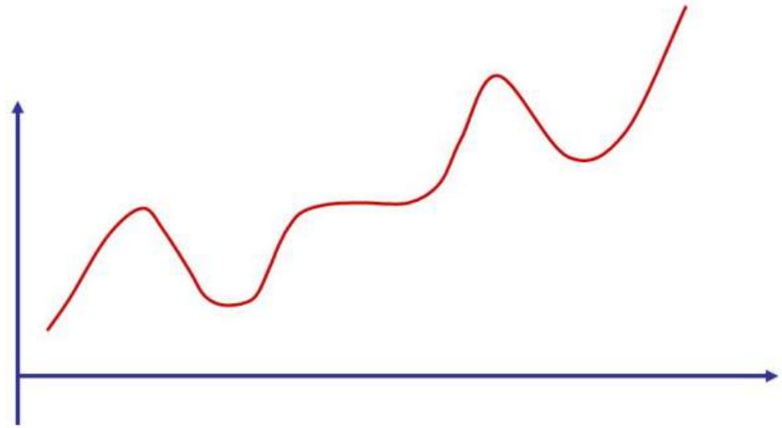
Calculate the error for each instance and then update the weights



CONVEX AND NON CONVEX



Convex



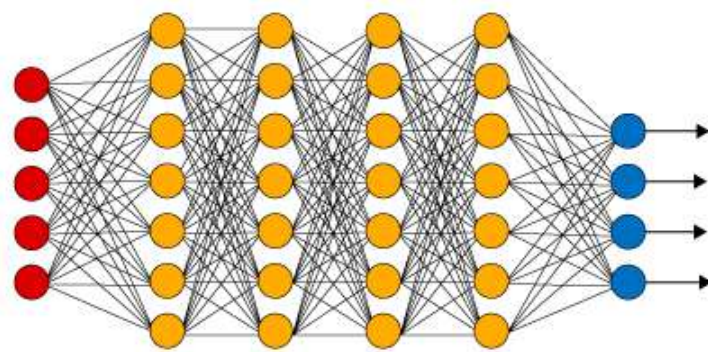
Non convex

GRADIENT DESCENT

- Stochastic gradient descent
  - Prevent local minimums (non convex)
  - Faster
- Mini batch gradient descent
  - Select a pre-defined number of instances in order to calculate the error and update the weights

DEEP LEARNING

- 90’s: SVM (Support Vector Machines)
- From 2006, several algorithms were created for training neural networks
- Two or more hidden layers



python

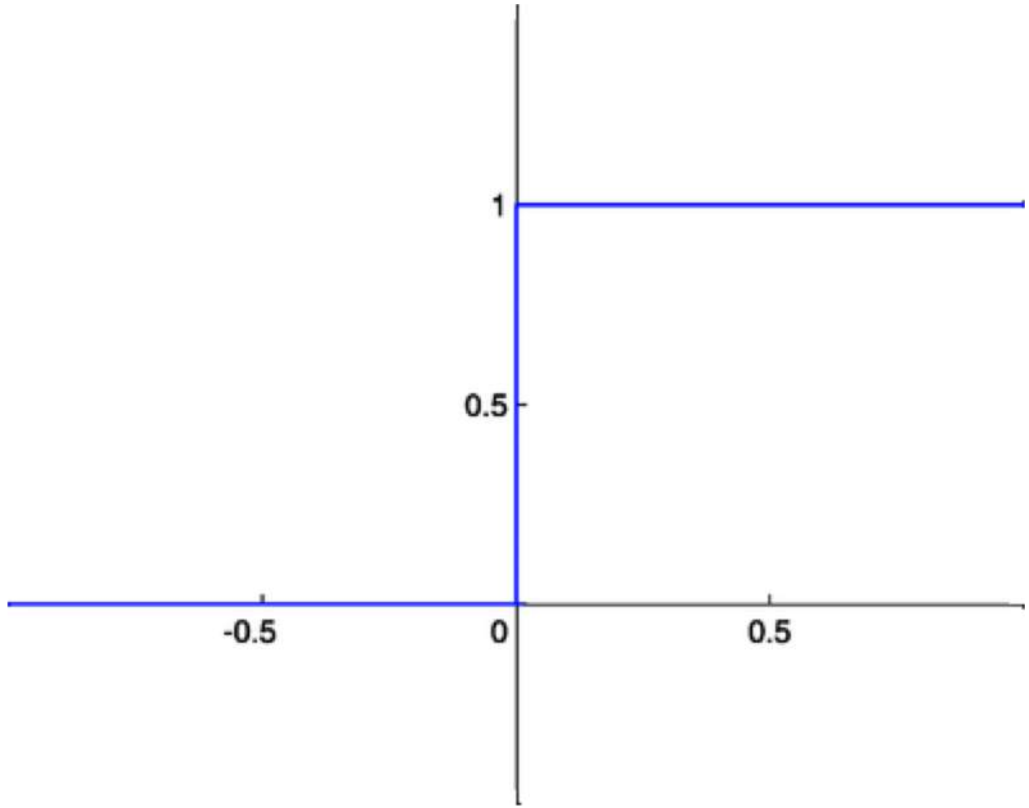
DEEP LEARNING

- Convolutional neural networks
- Recurrent neural networks
- Autoencoders
- GANs (Generative adversarial networks)

PLAN OF ATTACK – LIBRARIES FOR NEURAL NETWORKS

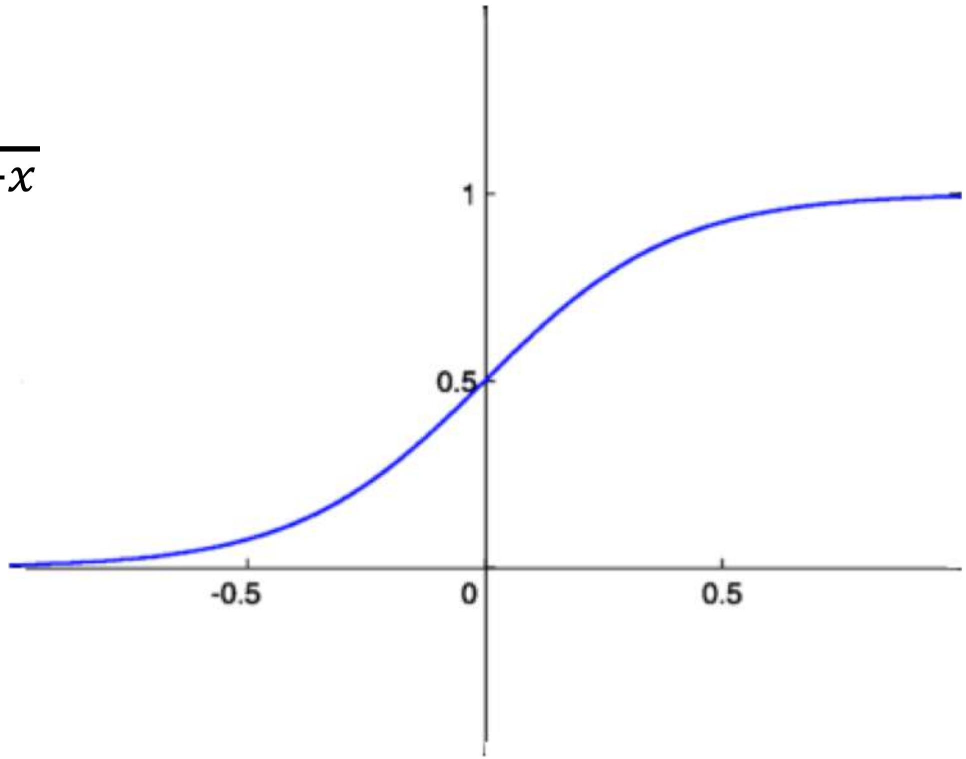
1. Pybrain
2. Sklearn (classification and regression)
3. TensorFlow (image classification)
4. PyTorch

STEP FUNCTION



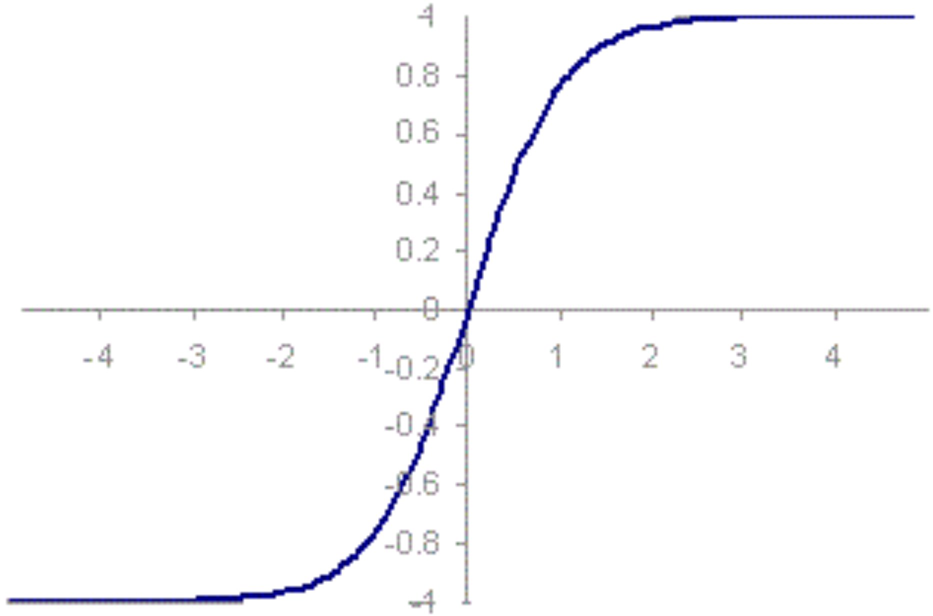
SIGMOID FUNCTION

$$y = \frac{1}{1 + e^{-x}}$$



HYPERBOLIC TANGENT

$$Y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

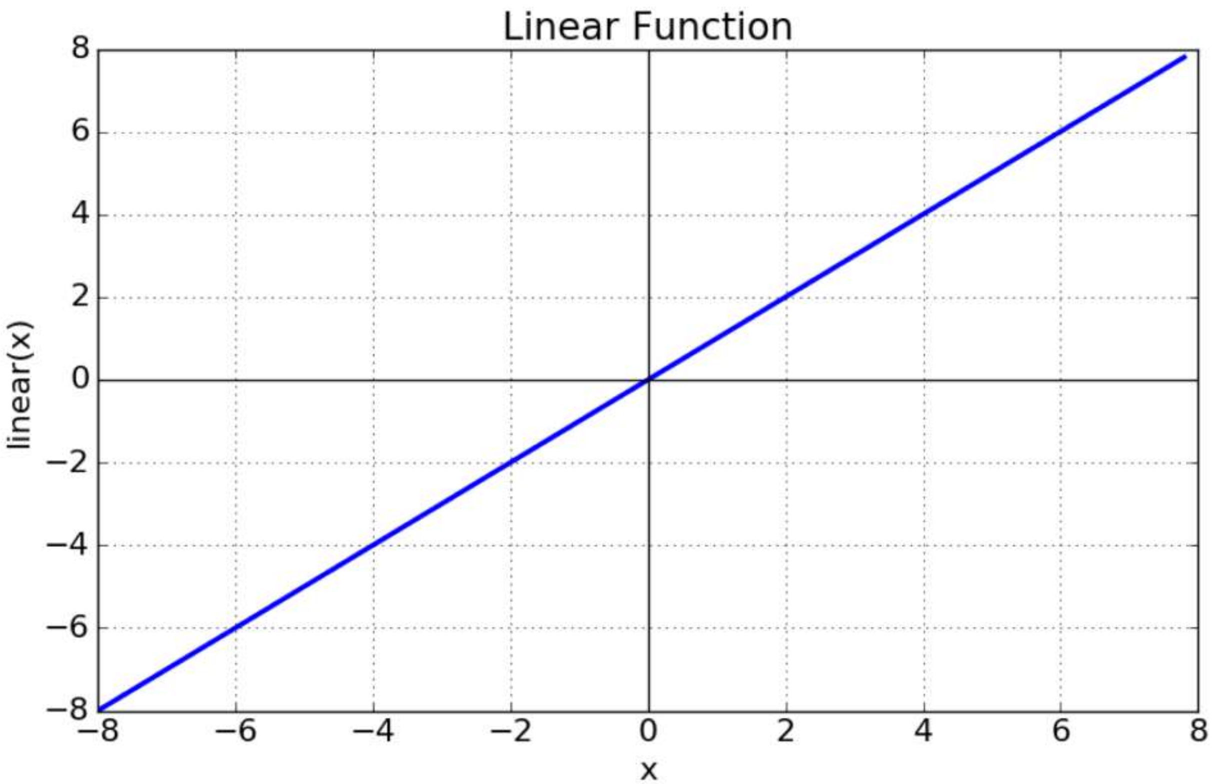


python

ReLU (RECTIFIED LINEAR UNIT)

$$Y = \max(0, x)$$

LINEAR



python

SOFTMAX

$$Y = \frac{e(x)}{\sum \quad e(x)}$$